REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	OCT. 30, 1993	FINAL TECHNICAL REPORT

4. TITLE AND SUBTITLE
THE IMPACT OF ORGANIZATION STRUCTURE ON TEAM DECISION MAKING UNDER STRESS AND UNCERTAINTY

5. FUNDING NUMBERS
Grant Number
N00014-90-J-1680

R & T Project Numb
URI 5202-9003

6. AUTHOR(S)
Paul E. Lehner and Alexander H. Levis

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Center of Excellence in Command, Control, Communications and Intelligence
George Mason University
Fairfax, VA 22030

8. PERFORMING ORGANIZATION REPORT NUMBER
GMU/C^{3}I-144-R

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
Arlington, VA 22217-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unlimited.

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The objective of the research reported herein is to investigate coordination in team decision making. Particular focus is placed on the identification and characterization of variables that enhance coordination and enable teams to maintain coordinated action under stressful conditions characteristic of tactical environments.

19960426 078

14. SUBJECT TERMS	15. NUMBER OF PAGES
Team Decision Making - Distributed Intelligence Systems - Belief Networks - Petri Nets	142
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified.	Unclassified.	Unclassified.	UL

CENTER OF EXCELLENCE IN
COMMAND, CONTROL, COMMUNICATIONS AND INTELLIGENCE

GEORGE MASON UNIVERSITY
Fairfax, Virginia 22030

FINAL TECHNICAL REPORT

for the period

16 March 1990 - 15 March 1993

for

THE IMPACT OF ORGANIZATION STRUCTURE ON TEAM DECISION MAKING UNDER STRESS AND UNCERTAINTY

Grant Number N00014-90-J-1680
R&T Project Number URI 5202-9003

Submitted to:
Dr. W. S. Vaughan, Jr. (3 copies)
Office of Naval Research
800 North Quincy Street
Arlington, Virginia 22217-5000

Copies to:
Director, Naval Research Laboratory
Administrative Grants Office, ONR
Defense Technical Information Center

Submitted by:

Paul E. Lehner
Alexander H. Levis
Co-Principal Investigators

October 30, 1993
Report #: GMU/C3I-144-R

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 PROGRAM OBJECTIVES

The objective of this research was to investigate coordination in hierarchical team decision making. Particular focus was placed on the identification and characterization of variables that enhance coordination and enable teams to maintain coordinated action under stressful conditions characteristic of naval tactical environments.

1.2 RESEARCH PLAN

The research plan described our strategy for meeting the program objectives and fulfilling the project tasks. Specifically, the research plan identified a series of research tasks. As documented in the quarterly progress reports, the research plan evolved several times during the duration of this effort.

The plan for the first two years of this research program was organized into three highly related research areas:

(a) Analytical models of C3I organizations that incorporate coordination variables;
(b) Descriptive models of team decision making; and
(c) Prescriptive models of team decision procedures.

As a result of progress in these areas, a fourth research area was introduced that integrated the approaches in areas (a) and (c):

(d) Prescriptive models of adaptive C2 organizations.

This area was a natural evolution of the research program and represented an effort to merge together results from the cognitive and the engineering aspects of the research; this is the natural next step towards the development of a theory of C2 organization design that encompasses both fixed and variable structures.

Each of these areas is discussed briefly below. A detailed discussion of the research accomplishments is provided in Chapter II to IV.

The focus of the first area was the development of methodologies, models, theories and algorithms directed toward the derivation of tactical decision, coordination, and communication strategies of agents in organizational structures. Both fixed and variable organizational structures were considered. However, the focus was on modeling variable organizational structures and how those structures adapt under conditions of stress. The framework for this research is analytic. The following tasks addressed this research area:

(1) Coordination in Decision Making Organizations
(2) Design of Multilevel Hierarchical Organizations.

The focus of the second area was the development of descriptive models of human decision making that are relevant to predicting team decision making performance under stress. For this work, it was assumed that the team members are well-trained. Consequently, the focus of the research was to identify conditions under which team performance degrades because one or more team members cannot effectively execute trained procedures properly. The following tasks addressed this research area:

(3) Experimental Research to Evaluate Vulnerable-to-bias Decision Procedures
(4) Quantitative Models of Combined User/Decision Aid Performance.

The focus of the third area was to develop a prescriptive methodology for specifying team decision making procedures. This work combined the normative and descriptive research in the first two areas to develop a methodology for deriving a set of robust team decision procedures. This includes procedures for coordinating team decision making activities and adaptation of coordination procedures. The following two tasks were carried out.

(5) Methodology for Prescribing Team Decision Procedures
(6) Automated Tools for Specifying Decision Procedures

The fourth area reflects an integration of the results obtained in the previous three research areas. Specifically, this research area addressed the problem of developing an integrated procedure that moves from an initial prescription of decision making procedures (research area c) to a detailed analytic model of the organization (research area a). To address this problem, that following research tasks were carried out:

THIS
PAGE
IS
MISSING
IN
ORIGINAL
DOCUMENT

CHAPTER II

ON A METHODOLOGY FOR TEAM DESIGN USING INFLUENCE DIAGRAMS

2.1 INTRODUCTION

A *team* may be defined as a group of experts, with overlapping areas of expertise, that work cooperatively to solve decision problems. Command and control (C2) teams are a specific class of teams where:

(a) each team member is responsible for an assigned set of tasks,

(b) each team member is well trained for the tasks that he or she is to perform,

(c) the team members have the common goal of satisfactorily solving the decision problems,

(d) the decision problems addressed by the team are often severely time constrained, and

(e) inappropriate decisions may lead to catastrophic consequences.

In many organizations, the behavior of the teams within that organization is guided by a series of policy and procedure rules. Consider, for instance, air traffic control systems. The behavior of a ground control team is guided in large measure by a set of procedural rules that specify how the team should react to various circumstances. The procedure rules specify conditions for de-icing, rerouting, priorities for landing, etc. Policy rules, in turn, provide guidelines for the establishment of the procedure rules (e.g., In snow, aircraft should be de-iced no less than one half hour before takeoff.)

A proposed set of rules for governing an organization's behavior can be evaluated in several different ways. One way is to evaluate them in terms of their logical *consistency* and *completeness*. Do the rules always result in a consistent recommendation, or can different rule subsets lead to different actions? Do they specify what to do under all circumstances? Alternatively, rules can be evaluated in terms of their *executability*. Although a rule set may be internally consistent, it may be difficult to define an acceptable architecture that can execute those rules (e.g., an architecture with efficient key threads.)

Finally, one can look at the expected *performance* of a proposed rule set. It is generally recognized that performance evaluation presupposes a model of the set of decision situations that a team will face are specified. Consider, for instance, the issue of whether a hierarchical or distributed team structure is more effective. Clearly this depends on the situations the team will face. If the decision situations evolve rapidly, and there are many decision tasks that can be handled independently, then a distributed structure will do better. On the other hand, if the situations evolve slowly, and individual tasks need to be coordinated, then hierarchical control will do better. This suggests that the very concept of a well-designed team presupposes a *domain model* - a description of the decision situations.

The basic philosophy behind the methodology is that domain models are not only useful for evaluating team decision procedures, but that they can also be used to *derive* team procedures. Specifically, given a domain model, one can derive team decision procedures that can be reliably executed by human team members and are nearly optimal in performance.

The methodology described in this chapter aims at specifying the decision procedures of a team that addresses a repetitive or ongoing decision problem and at generating a team architecture that includes:

(a) a well-defined team structure that identifies the functions each team member should perform and the communication paths between team members, and

(b) an assignment of well-defined tasks to each team member

The methodology consists of the following steps:

- Develop the domain model in form of an influence diagram.
- Derive the Team Decision Procedures in form of a default tree (DTree).
- For each path on the DTree from the root to a leaf, generate the set of fixed structures. A fixed structure is a Petri Net describing the sequence of items of evidence examined in this path.
- Derive a set of candidate Functional Architectures. Each candidate is a Colored Petri Net (CPN) obtained by picking one member from each set of fixed structures and folding it with the others.
- Check coordination constraints on the different members of the set to discard infeasible ones.
- Map the set of candidate functional architectures onto a Physical Architecture to obtain a set of candidate Operational Architectures representing the team procedures. Performance Evaluation can be conducted on those candidates for the final selection.

2.2 DEVELOP DOMAIN MODEL

The initial step of the methodology is to develop a domain model. Domain models are represented using a decision theory formalism called an *influence diagram* or a *decision network.*[1] A decision network provides both a graphic and mathematical description of the probabilistic relationships between objects, events, and decisions in a problem domain, as well an assignment of utilities to various outcomes (Neapolitan, 1990). Over the last decade, decision networks have emerged as a standard tool for eliciting and encoding probability and utility information. If properly structured, they minimize the number of probability and utility assessments required.

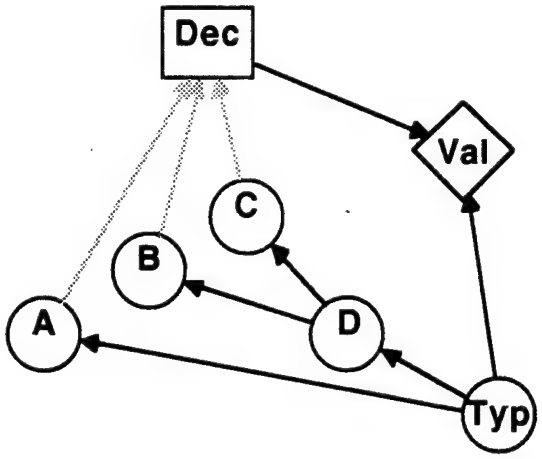


Figure 2.1 An Influence Diagram

A simple example of a decision network is shown in Figure 2.1. The circle nodes are *chance nodes*. Each chance node identifies a set of mutually exclusive and exhaustive propositions. The arcs between chance nodes identify the conditional probability statements that must be contained in each node. For instance, the node Typ contains the unconditional probability distribution P(Typ). The node A contains the conditional probability distribution P(A|Typ). The rectangular node is a *decision node.* Each decision node identifies an exhaustive set of mutually exclusive decisions. Arcs going from a chance node to a decision node are information arcs. They identify information that will be available when the decision must be made. The diamond node is a *value node.* A value node assigns a utility to each row in the cross product of the propositions/decisions of its parent nodes. For instance, the node Val in Figure 2.1 assigns a utility for each possible decision in Dec and state in Typ.

---

[1] Influence diagrams and decision networks are also referred to as decision graphs. Also, belief networks, Bayesian belief networks, and inference networks are specialized types of influence diagrams.

Chance nodes that have information arcs going to a decision node are referred to as *evidence items*. An evidence item is *instantiated* when the value of that evidence item is known. For instance, A = a1 asserts that the *evidence value* for evidence item A is a1. A set of evidence values for all evidence items is referred to as an *evidence state*. For instance, the vector <a1, b2, c3> describes the evidence state where A = a1, B = b2 and C = c3.

Once a decision network has been defined, there are a variety of algorithms and software tools for processing the network (Buede, 1992). These algorithms can be used to derive the expected utility of any decision, or the posterior probability of any chance node conditioned on specific values for any subset of the chance nodes. Consequently, they can be used to evaluate the performance (in terms of utility) of *any* decision procedure.

2.3 DERIVE TEAM DECISION PROCEDURE

The second step is to derive an overall team procedure. Two general considerations are relevant here - performance and bounded rationality. If bounded rationality was not a consideration, then one could simply require that the team uses the domain model to derive the maximum expect utility choice for each input situation. Of course, this is humanly impossible. Consequently, one must either allocate the decision processing to a computer or find decision procedures that are easier to execute, but approximate the performance of expected utility maximization. For those decisions that we are unwilling to allocate to machines, it is necessary *compile the domain model* into a set of humanly-executable decision procedures.

Lehner and Sadigh (1993) have developed several algorithms for compiling domain models into simple decision procedures. These algorithms will compile an influence diagram into a structure called a default tree (DTree). A DTree is composed of default nodes (Dnodes) and evidence nodes (Enodes). Each Dnode specifies a decision, while each Enode specifies both an evidence item and a decision. To illustrate, the DTree in Figure 2. 2 contains 4 Enodes and 6 Dnodes. This DTree corresponds to a decision procedure which begins by either selecting d1 or examining evidence item A. If A is examined and its value is a2, then the decision d1 is immediately selected. If A = a1, then d2 is selected or B is examined. If B = b1, then the decision d1 is selected, else if B = b2 then d2 is selected. Returning to the root Enode, if A = a3 then select d3 or examine C. If C = c1, then select d1. If C = c2 then select d3 or examine B. If B = b1, then select d2, otherwise select d3.

THIS

PAGE

IS

MISSING

IN

ORIGINAL

DOCUMENT

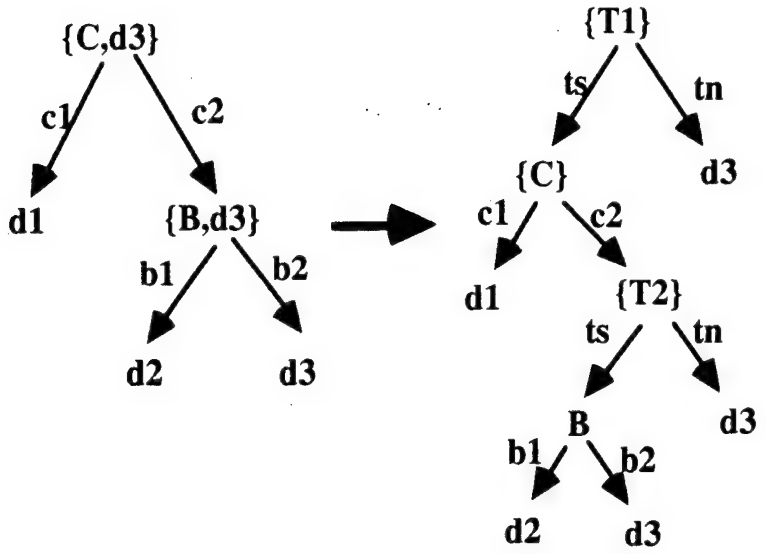


Figure 2.3 Deterministic DTree Derivation

In a deterministic DTree, each root or subroot corresponds to a item of evidence to be examined. According to the value of this evidence, another item of evidence is examined until a conclusion (a "leaf" of the tree) is reached. The deterministic DTree provides in fact an answer to the following question: according to the current item of evidence, what is the next item of evidence which needs to be looked at? This means that to reach a conclusion, it may not be necessary to look at all the items of evidence. The item of evidence at the root of the tree will always be looked at first because it represents the one bringing the highest utility to the organization. An example is given in Figure 2.4. The organization has to decide whether a specified object is an Enemy (En), a Friend (Fr) or Neutral (Ne). To reach the conclusion, three different items of evidence can be looked at: (1) R (for example, Radar) that can take values R1, R2, or R3, (2) P (for example, passive sensor) that can take values P1 or P2, and (3) Q (for example, report from intelligence) that can take values Q1, Q2 or Q3. An input to the organization is represented by a 3-tuple (r, p, q) where r (resp. p, q) is the value of the item of evidence R (resp. P, Q). Let assume that the input to the organization is (R3, P2, Q2). Item of evidence R is at the root of the tree and is examined first. Its value is R3 and according to the DTree, the next item of evidence to look at is Q. Its value is Q2 and the conclusion that the object is an Enemy can be reached. To reach this conclusion, the item of evidence P has not been examined.

THIS
PAGE
IS
MISSING
IN
ORIGINAL
DOCUMENT

stage model is considered: Situation Assessment, Information Fusion (this stage aggregates the IF, TP and CI stages of the five stage model) and Response Selection

We assume that each FDMU knows initially (at its SA stage) only the value of the item of evidence that is assigned to it. Then, it receives in its IF stage the values of other items of evidence from other FDMU. In its RS stage, it decides if a conclusion can be reached or if it has to send its evidence set to the FDMU in charge of the next item of evidence to look at (as specified by the DTree). A FDMU can be represented by a Petri Net as shown in Figure 2.5.

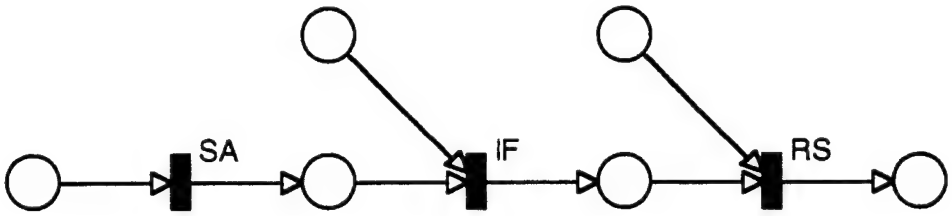


Figure 2.5 Three Stage Model of a FDMU

Each item of evidence represents an input to the organization; each value taken by the different items of evidence induces a path in the DTree from the root to one of the leaves. Each path to a conclusion defines therefore a specific set of interactions between the different FDMU that leads to a conclusion. A fixed functional structure can then be defined for each path from the root to a leaf of the tree.

Two approaches were considered for generating the set of fixed structures :

- Use of the Lattice Algorithm (Remy and Levis, 1988);
- Direct derivation of fixed structures.

2.4.1 The Lattice Algorithm

The Lattice Algorithm allows for the automatic generation of candidate architectures satisfying a set of connection constraints and of structural constraints. The candidate architectures (which are numerous) are not listed singly but gathered in lattices defined by their maximal and their minimal elements: the maximally and minimally connected architectures (MINOs and MAXOs). Each candidate architecture belongs to one of the lattices and can be defined by adding simple paths to the MINO or subtracting simple paths from the MAXO. The initial implementation of the algorithm by Remy (1986) was limited to five decision making units.

The input is a set of connection constraints represented by matrices: **e** (input to FDMU), **s** (FDMU to output), **F** (SA to IF stages), **G** (RS to SA stages), **H** (RS to IF stages), and **C** (RS to CI stages). The user enters in each cell of the different matrices either "1" if he wants the corresponding connection to exist, "0" if he does not, or "2" if the corresponding link is optional. From the interaction constraints entered by the user in the different matrices, the algorithm generates two Petri Nets: the Kernel Net corresponding to the structure deduced from the connection matrices where all the optional links are considered inactive and the Universal Net corresponding to the structures where all optional links are considered active. In a second stage, the algorithm computes the S-invariants of the Universal Net and stores those that contain the source place and the sink place (the simple paths). In a third stage, MINOs are generated by adding simple paths stored in the second stage to the Kernel Net until all the structural constraints are satisfied. Remy (1986) has defined four basic structural constraints:

- R1 The structure should be connected
- R2 The structure is acyclical (no loop)
- R3 There exists at most one link from the RS stage of DM_i to the SA, IF, CI stages of DM_j: $\mathbf{G}_{ij} + \mathbf{H}_{ij} + \mathbf{C}_{ij} \leq 1$
- R4 Information Fusion takes place only at the IF and CI stages. The SA stage has at most one input with preference to the external input: $\mathbf{e}_j + \mathbf{G}_{ij} \leq 1$

In a fourth stage, MAXOs are generated by removing simple paths from the Universal Net until none of the constraints are violated. Finally, the algorithm computes the invariants of the different MINOs and MAXOs to check the absence of loops. If there is a loop, the corresponding MINO or MAXO is discarded.

In this problem, the number of FDMUs is the number of items of evidence considered in the path from the root to a leaf of the DTree. All the FDMUs receive an input, therefore all the cells of **e** contain "1". To leave complete flexibility on which FDMU generates the output of the organization, "2" is assigned to each cell of **s**. On the other hand, the CI stage has been suppressed in the FDMU; therefore, all the cells of **C** contains 0. Finally, since the SA stage of a FDMU can receive only one input (because of the structural constraint R4) and that input is from a source, there can not be any connection between the RS stage of a FDMU and the SA stage of another FDMU: the cells of matrix **G** contain "0". To represent the sequence of items of evidence examined in a path from the root to a leaf of the DTree, the following rules are applied:

- If FDMU1 is responsible for examining the item of evidence at the root of the DTree and FDMUi is responsible for examining the next item of evidence in the path, the cell $\mathbf{F}_{1i}$ has to contain "1". The other cells of matrix F contain "2",
- If FDMUi and FDMUj are responsible for two successive items of evidence in the sequence represented in the path, the cell $\mathbf{H}_{ij}$ has to contain "1". Once all the successive items have been considered, the remaining cells of matrix H are assigned the value "2".

For the example described earlier, let us generate a functional fixed structure for the input (R3, Q1, P1). Only items of evidence R3 and Q1 are used to reach the conclusion and therefore only two FDMUs need to be considered. The set of connection constraints could be as shown in Figure 2.6.

e: ext → SA

R	Q
1	1

s: RS → ext

R	Q
2	2

F: SA → IF

	R	Q
R	X	1
Q	2	X

G: RS → SA

	R	Q
R	X	0
Q	0	X

H: RS → IF

	R	Q
R	X	2
Q	2	X

C: RS → CI

	R	Q
R	X	0
Q	0	X

Figure 2.6 Connection Matrices for Input (R3, P1, Q1/Q2/Q3)

Running the Lattice Algorithm for this set of connection constraints shows that all the feasible fixed structures are gathered in lattices with two MINOs and two MAXOs. The Petri Nets are displayed in Figure 2.7.

All feasible fixed structures are obtained by adding Simple Paths to either of the MINOs. In this example, there are twenty feasible structures to choose from. The selection is made according to some criteria (degree of redundancy, or the number of inter-FDMU exchanges, for example) but as the number of items of evidence increases, the number of inputs increases and the number of feasible structures for a given input increases exponentially. The architecture designer might have to add connection constraints to reduce the number of solutions.

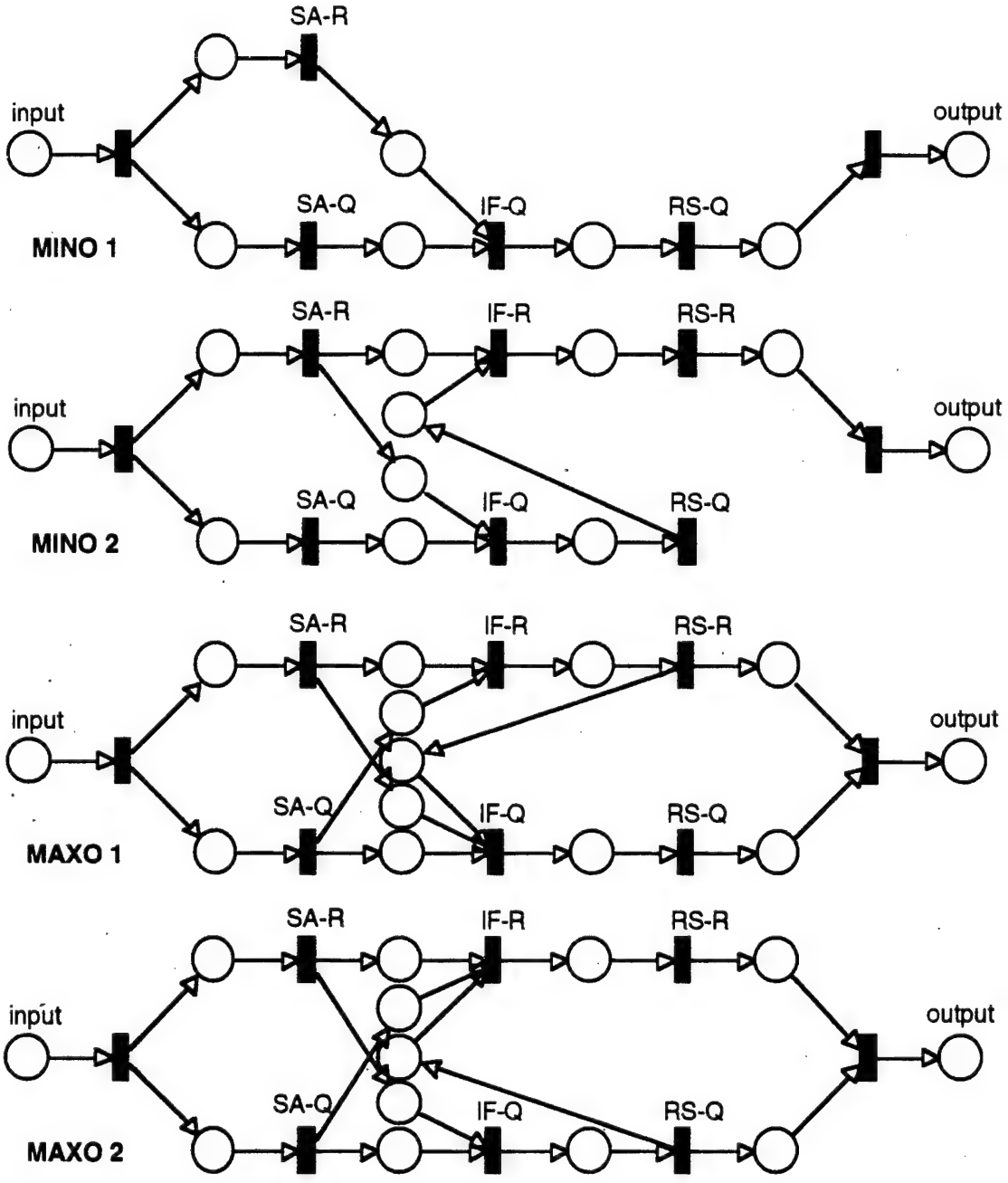
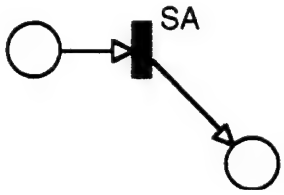


Figure 2.7 MINOs and MAXOs for Input (R3,P1,Q1/Q2/Q3)

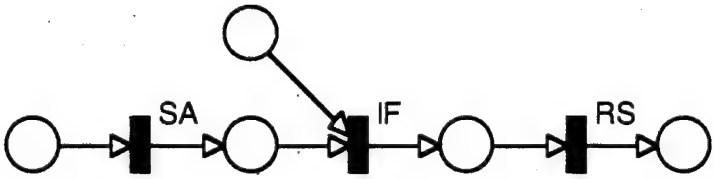
THIS
PAGE
IS
MISSING
IN
ORIGINAL
DOCUMENT

satisfy the structural constraints. There is a need then to determine which redundant interactions has to remain in the set of fixed structure for this specific path on the DTree. For the example, the connection constraints are shown on Figure 2.10. Running the Lattice Algorithm on this set of constraints leads to one MINO (the direct derived structure) and one MAXO displayed in Figure 2.11. This MAXO includes additional interactions that enhance survivability.

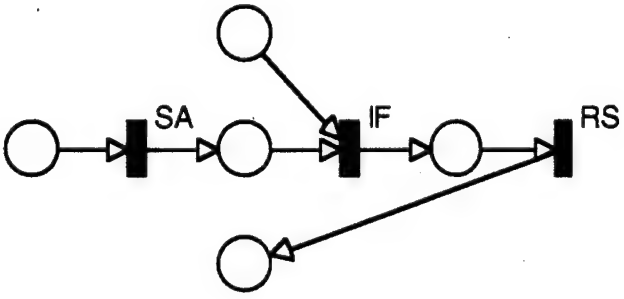
Root FDMU producing a conclusion



Root FDMU unable to produce a conclusion and sending its information to another FDMU



Non-root FDMU producing a conclusion



Non -root FDMU unable to produce a conclusion and sending its information to another FDMU

Figure 2.8 Different Kinds of FDMUs

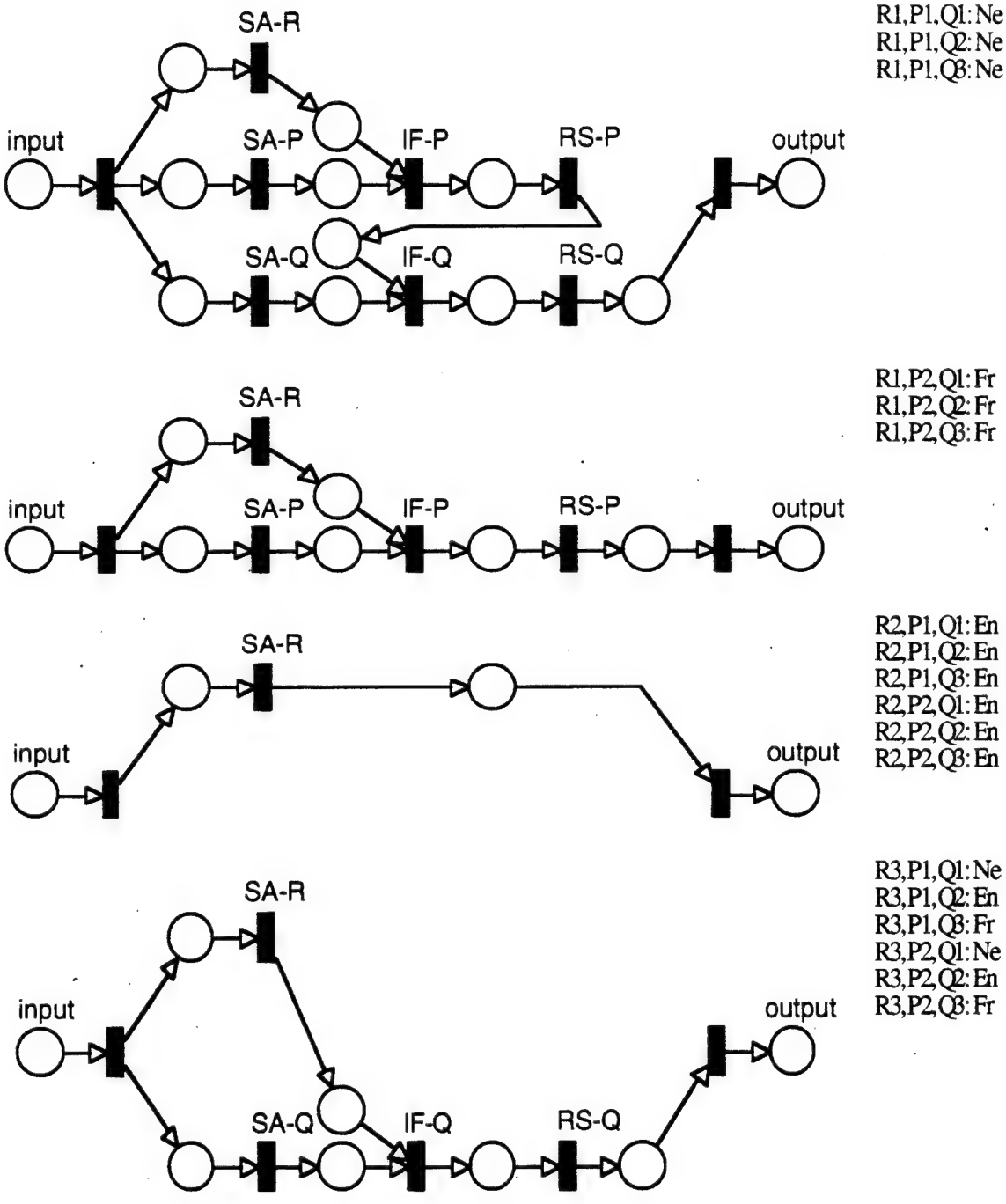


Figure 2.9 Set of Fixed Structure for the Example

e

ext → SA

R	Q
1	1

s

RS → ext

R	Q
1	0

F

SA → IF

	R	Q
R	X	1
Q	2	X

G

RS → SA

	R	Q
R	X	0
Q	0	X

H

RS → IF

	R	Q
R	X	2
Q	2	X

C

RS → CI

	R	Q
R	X	0
Q	0	X

Figure 2.10 Connection Matrices for Input (R3, P1, Q1/2/3) to Generate Superstructures of the Direct Derived Structure

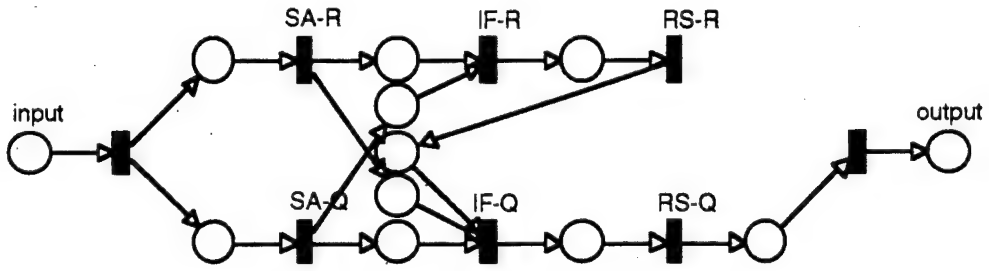


Figure 2.11 MAXO of the Superstructures of the Direct Derived Structure

2.5 FOLDING OF FIXED STRUCTURES FOR CANDIDATE FUNCTIONAL ARCHITECTURES DERIVATION

Once all the fixed structures for every path in the DTree are constructed, one has to pick one member of each set and fold with the others into a single structure depicting the functionality of the entire tree. The structure so obtained is a variable structure. Let us recall that the Fixed Structures were represented as Ordinary Petri Nets. Ordinary Petri Nets can be simulated with tokens that are undistinguishable. Since a fixed structure represents the sequence of processes that take place for a single event type, Ordinary Petri Nets were sufficient to represent these structures. In addition, the mathematical description of Ordinary Petri Nets is sufficient for the Lattice Algorithm. When the fixed structures are folded together, the events must retain their identity and be distinguishable.

Colored Petri Nets allow to do so. Colored Petri Nets are an extension of Ordinary Petri Nets in which tokens are no longer indistinguishable, but have attributtes (colors). A token is an element of a given color set and can only take the values defined by the color set. A place can only contain tokens of a given color set. Arcs are annotated to specify the number and the color of tokens that that the arc can carry. This arc expression can contain "if-then-else" or "case" statements to model alternate behaviors of a transition according to different combinations of tokens present in its input places. Colored Petri Nets are therefore suitable to represent the variable structures obtained by folding together fixed structures.

A number of candidate variable structures can be obtained as a result of different folding patterns. The most intuitive one is to superimpose all FDMUs responsible for the same item of evidence across the different fixed structures. The folded Colored Petri Net for the example is shown in Figure 2.12. Note that the transitions are represented now by boxes with inscriptions.

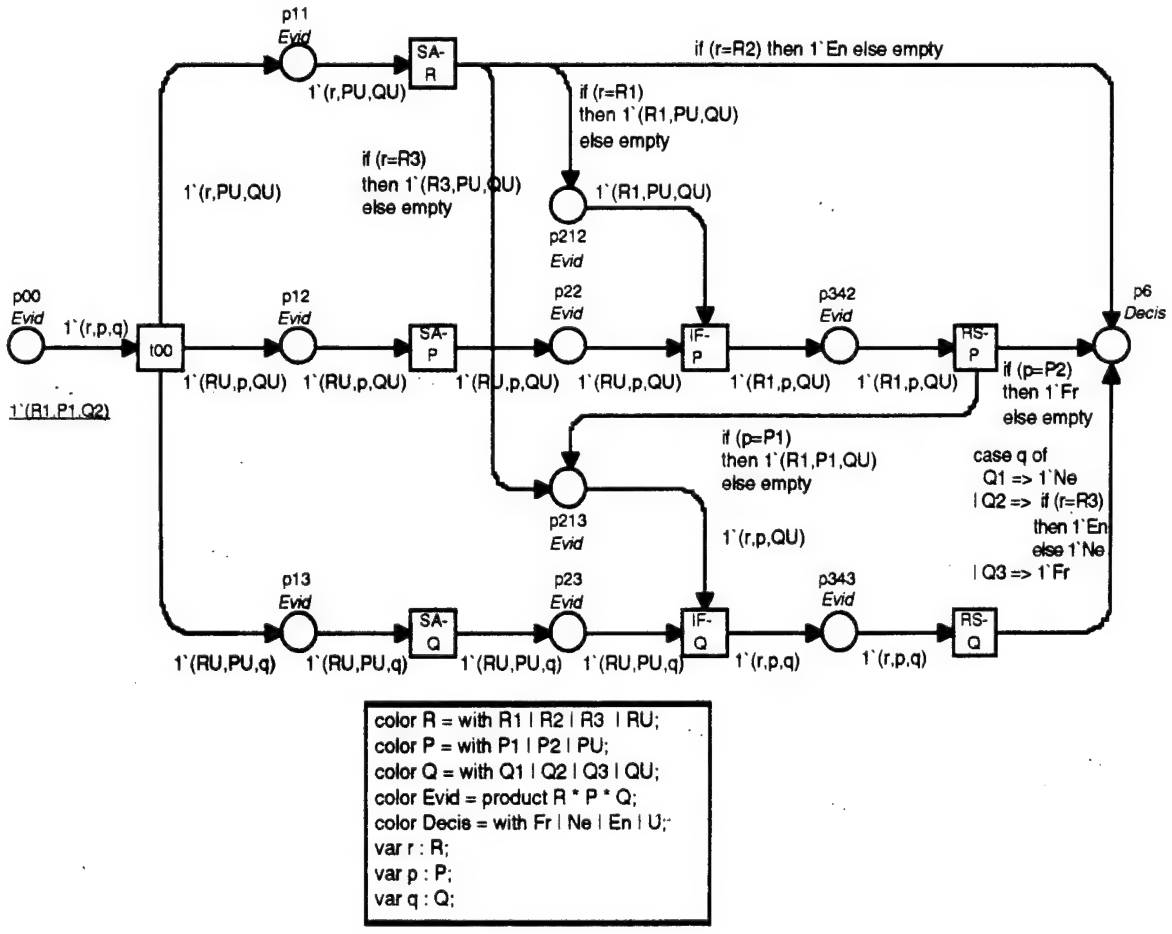


Figure 2.12 Folded CPN for the Example

2.6 COORDINATION CONSTRAINTS

As mentioned in the previous section, fixed structures are generated for each path from the root to the terminating leaves of the DTree. The set of fixed structures so constructed represents the feasible Functional Architectures required to carry out a particular task /rule described by the path of the DTree. In order to derive a set of candidate Functional Architectures for the entire DTree, one member from each set of fixed structures is picked and folded into a single Colored Petri Net (variable) structure. The process is repeated until all such variable structures are constructed. But not all folded structures so obtained represent *feasible* Functional Architectures. In order to be a feasible variable structure, a folded structure must satisfy the *Coordination Constraint.* A detailed description of the coordination constraint is given in Chapter IV.

A component (represented as a transition in Petri Net formalism) is said to meet the Coordination Constraint if it is not required to respond differently for inputs which are indistinguishable to the task. A variable structure is said to meet the Coordination Constraint if all its components meet the Coordination Constraint. A structure satisfying the constraint is feasible in the sense that it can be realized, and consequently can be mapped on a Physical Architecture.

The need for checking variable structures obtained by folding ordinary Petri Net structures was first pointed out by Demaël (1989). The problem presented by Demaël was formulated analytically by Lu and Levis (1992). They proposed an algorithm which, given a set of fixed structures for each input situation (represented by the item of evidences inducing a single path in the DTree) and a folding strategy, checks each component of the folded net to determine whether it satisfies Coordination Constraint. Once all components are found to satisfy the constraint, the folded structure associated with the folding scheme is declared feasible. For a detailed and rigorous discussion on the algorithm, see Lu (1992). The following paragraph presents an illustration of the idea.

The Colored Petri Net of Figure 2.12 represents a feasible variable structure since all the transitions in the structure meet the Coordination Constraint; none of them is required to respond differently for input situations indistinguishable to it. However, if the first fixed structure represented by the Ordinary Petri Net of Figure 2.9 is replaced by the one of Figure 2.13 and the same folding strategy is employed, then the variable structure represented by the Colored Petri Net of Figure 2.14 (with annotations suppressed) is obtained. Note that the structure in Figure 2.13 is a feasible fixed structure; however, the variable structure obtained in Figure 2.14 does not satisfy the Coordination Constraint. The argument on infeasibility is given as follows:

The transition SA-R in structure of Figure 2.13 is required to send its assessed information to IF-P and IF-Q whenever it gets R1 as the input item of evidence. Note (by definition) that the transition SA-R can only distinguish among the elements of the alphabet R, it can not distinguish P1 from P2 and so on. However, the same transition SA-R in the second fixed structure of Figure 2.9 is required to send information only to IF-P under the inputs that are identical to the inputs in the previous case if seen by SA-R , thus introducing a requirement that can not be satisfied by the transition. Therefore, if these fixed structures are folded together in a manner that all SA-R stages are folded into a single transition in the variable structure, the resulting variable structure will be infeasible and can not be realized.

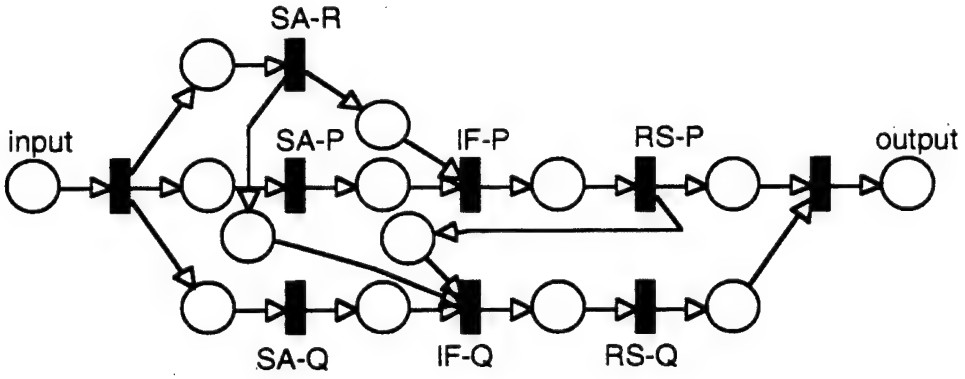


Figure 2.13 Fixed Structure for Inputs (R1, P1, Q1/Q2/Q3)

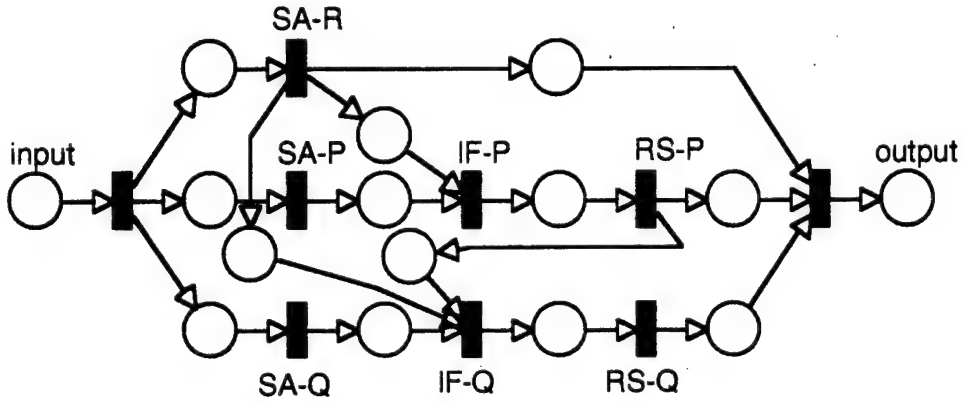


Figure 2.14 Infeasible Variable Structure

2.7 OPERATIONAL ARCHITECTURE DERIVATION

Once a variable structure is selected, the functional architecture is converted into the operational architecture, which is also represented as a Colored Petri Net, as follows.

Not all input places in the variable structure represent physical sources of information, i.e., sensors. As mentioned earlier, the output of a sensor can be a vector, where each element/ attribute takes the value from an evidence alphabet; the alphabet of the sensor, in such case, is a set of different alphabets. In order to construct a physical architecture, the places associated with the alphabets, which in turn are the elements of an alphabet associated with a physical entity, are *compounded* together. Compounding refers to folding different parts of a single structure. Once all such places are compounded together, these places now represent the physical sources of information. Figure 2.15a shows the simplified Colored Petri Net representation of a variable structure with three input sources of evidences, P (passive), R1 (speed) and R2 (bearing), where R1 and R2 are the elements of the output vector from a sensor R (radar). Figure 2.15b shows the compounding of the two sources into one physical source R.

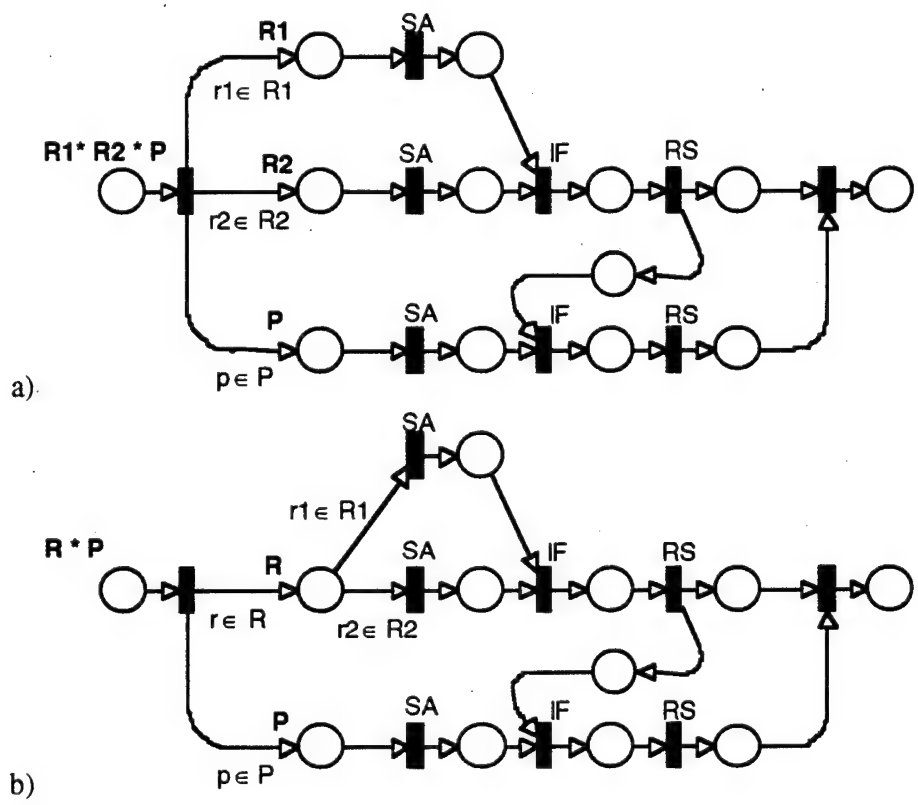


Figure 2.15 Compounding Input Sources

The processes obtained from the first step can now be assigned to different physical entities. At this point, processes in series can be assigned to a single physical entity. Function allocation across processes could only be done by folding.

An operational architecture represents the processes executed by the different team members and their interactions for the right execution of the mission assigned to the team. This operational architecture is derived from the functional architecture by defining the boundaries of the extent of responsibilities of each team member. In the example, a simple function allocation algorithm is to allocate the processes of each FDMU to a single human decision maker. The resulting operational architecture is shown on Figure 2.16 (arc annotations have been removed for clarity) and is derived from the variable structure of Figure 2.14 by representing by the means of round boxes the extent of responsibility of each team member.

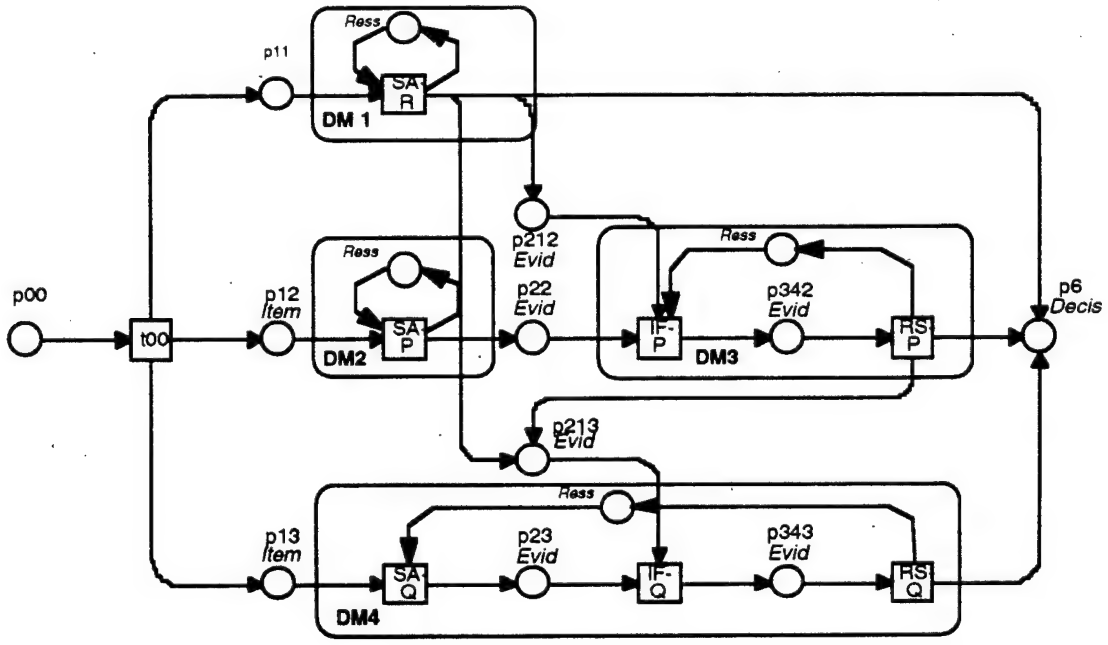


Figure 2.16 Operational Architecture for the Example

A number of function allocation algorithms can be applied at this point, where each algorithm is a function of a different parameter or a combination of parameters. Workload, available resources, multilevel relationships among organization members, the probability mass function of the DTree, are some examples of the parameters of the function allocation algorithms that can be considered. Section 2.8 describes a procedure to compute the workload associated with the algorithms to be used as the basis for allocation of functions to decision makers.

The use of different function algorithms leads to different candidate operational architectures that need to be evaluated for the selection of the most effective one. Each Colored Petri Net representation of the candidate operational architectures can be refined by adding resource places to each team member, showing the number of different inputs a decision maker can process simultaneously and by modifying the interaction links to include communications protocols and dynamic communication resources allocation. The different nets can then be simulated to show (1) that the DTree is correctly implemented and (2) to generate time-related measures of performance (response time, throughput) that can be used as the basis for the final selection of the operational architecture.

2.8 WORKLOAD EVALUATION FOR ALLOCATION OF FUNCTIONS TO DECISION MAKERS

Boettcher and Levis (1982) have developed an evaluation of the Human Decision Maker workload based on Information Theory. Using entropy H as the measure of the uncertainty in a random variable w_i that takes values w_{ij} and where:

$$H(w_i) = - \sum_j p(w_{ij}) \log p(w_{ij}) \qquad (1)$$

the total information processing activity of a decision maker, G may be expressed as:

$$G = \sum_i H(w_i) \qquad (2)$$

where w_i's are all the variables handled by the decision maker.

This total workload may be decomposed (Conant, 1976) in the following way:

$$G = G_t + G_b + G_c + G_n$$

The throughput or information transmission G_t is given by:

$$G_t = T(x:y) = H(y) - H_x(y)$$

where $H_x(y)$ is the conditional entropy of y given x.

The term G_c, the coordination is a measure of interconnectedness and may be expressed as:

$$G_c = T(w_1: w_2: \ldots : w_n :y)$$

The noise G_n is the entropy remaining in the internal variables and the output y when the input x is fully known:

$$G_n = Hx(w_1, w_2 ..., w_n , y)$$

Finally, the blockage G_b

$$G_b = T_y(x : w_1 : ... : w_n)$$

is the information in the input that is not in the output.

The application of information theory in the computation of human workload requires the assumption that the model is memoryless.

The bounded rationality represents the fact that the workload should not exceed some threshold so that rapid degradation of performance does not occur. It can be expressed in the following form:

$$G / \tau \leq F_0$$

that is, the information processing rate must be less than some threshold F_0 that depends on the task and the individual decision maker. This model of bounded rationality has been investigated experimentally by Louvet et al., (1988).

The workload induced by a function is computed as follows. The first step is to identify the variables of the function, their relationships and the values each of these variables can take. In a second step, for a scenario of a distribution of inputs, the distribution of the values taken by the variables is computed through simulation. Then the entropy of each variable is computed using equation (1). The total workload induced by the function is then evaluated using equation (2).

To make sure that the bounded rationality constraint is satisfied, one can compute, for a given scenario of inputs with a given distribution of occurrence, the workload induced by the different functions represented by transitions in the Colored Petri Net model of the functional architecture. The allocation strategy is therefore to assign functions to decision makers so that the total workload induced by the set of the different functions allocated to a decision maker satisfies the bounded rationality constraint. Instead of producing only one candidate architecture, this procedure will discard candidate architectures in which the bounded rationality constraint is violated for one or more decision makers.

It is important to point out that the total workload induced by two sequential functions is not equal to the sum of the workload induced by each function. The output of the first function which is the input of the second function becomes an internal variable whose entropy needs to be counted only once. Figure 2.17 illustrates this point. F1 and F2 are two sequential functions. F1 has x for input and produces the output x_{12}. F2 has for inputs x_{12} and z and produces the output y. If $H_{int}(F1)$ and $H_{int}(F2)$ are the sums of the entropies of the internal variables of F1 and F2, and W_{F1} and W_{F2} denote the workload induced by F1 and F2 for a distribution of inputs, we have:

$$W_{F1} = H(x) + H_{int}(F1) + H(x_{12})$$
$$W_{F2} = H(x_{12}) + H(z) + H_{int}(F2) + H(y)$$

If F1 and F2 are allocated to the same decision maker, the workload induced by these two functions, W_{F1+F2}, is :

$$W_{F1+F2} = H(x) + H_{int}(F1) + H(x_{12}) + H(z) + H_{int}(F2) + H(y)$$

and therefore:

$$W_{F1+F2} = W_{F1} + W_{F2} - H(x_{12})$$

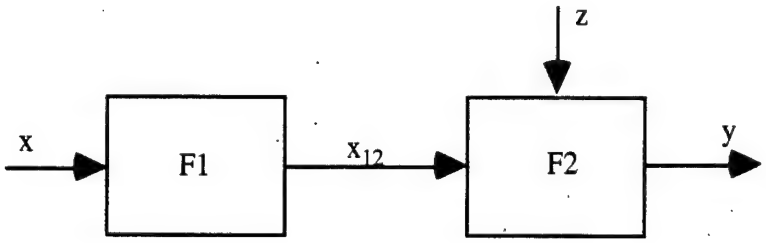


Figure 2.17 Two Sequential Functions F1 and F2

For problems of interest, the algorithms F1 and F2 are simple enough to compute the distributions of the variables handled by the decision makers. The formulas for each case are computed in the next sections.

2.8.1 Situation Assessment for the Evidence Item at the Root of the DTree

The generic algorithm for the SA stage for the item of evidence at the root of the tree is displayed on Figure 2.18. Starting with the input E1, a value is attributed to the variable C according to rule of the DTree. The value of E1 is transmitted to other $FDMU_i$ if the value of C is Ci, for $i \leq n$. If the value of C is Cn + 1, the decision of the organization is produced.

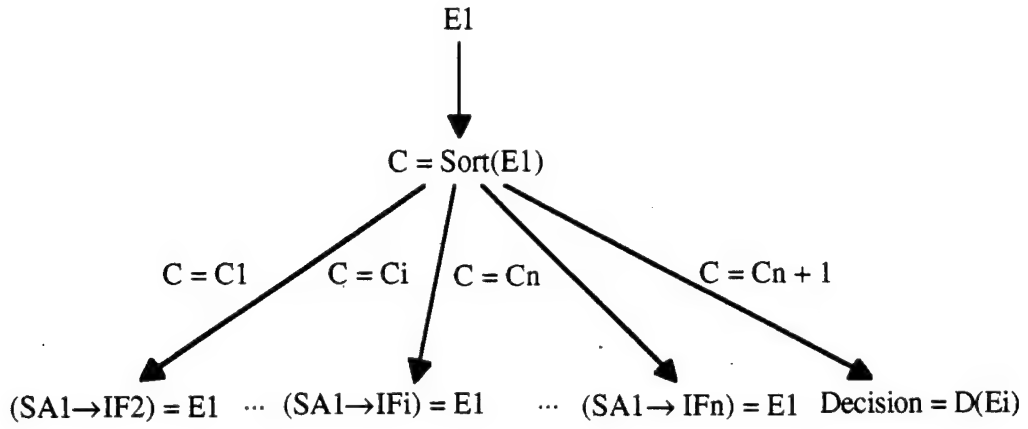


Figure 2.18 Generic Algorithm for the SA Stage for the Evidence Item at the Root of the DTree

The expression for the workload induced by the SA stage function, WSAroot is:

$$W_{SAroot} = H(E_1) + H(C) + \sum_i H(SA_1 \rightarrow IF_i) + H(Decision|E_1)$$

2.8.2 Situation Assessment for Evidence Items that are not at the Root of the DTree

The algorithms used in this case is the idenyity; it copies the data provided by the sensor as shown in Figure 2.19.

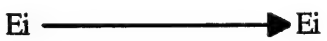


Figure 2.19 Generic Algorithm for the SA Stage for Evidence Items not at the Root of the DTree

The expression W_{SAi} for the workload induced by the SA stage function for items of evidence that are not at the root of the tree is therefore:

$$W_{SAi} = 2\ H(Ei)$$

2.8.3 Information Fusion

This algorithm combines pieces of evidence to build tuples as shown in Figure 2.20. The assumption is that there exists only one buffer in which all the information transmitted by other FDMUs is stored until processed. The tuple (E1, ..., Ei-1, Ei+1, ..., En) represents the information transmitted by other FDMUs. Some of the E_j's can have an unknown value

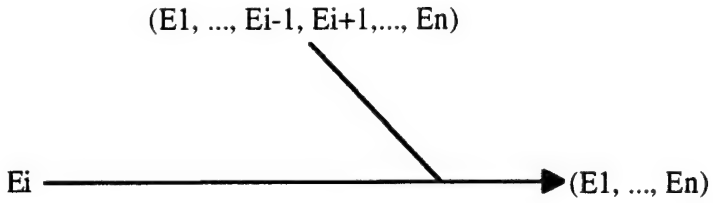


Figure 2.20 Generic Algorithm for the IF Stage

The expression for workload induced by the IF stage of the FDMU dealing with the evidence item Ei, W_{IFi}, is therefore:

$$WIFi = H(Ei) + H(E1, ..., Ei-1, Ei+1, ..., En) + H(E1, ..., En)$$

2.8.4 Response Selection

The generic algorithm for the RS stage for the item of evidence Ei is displayed in Figure 2.21. Given the input (E1, ..., En), it consists in attributing a value to the variable C according to rule of the D-Tree. The value of (E1, ..., En) is transmitted to other FDMUi if the value of C is Ci. If the value of C is Cn + 1, the decision of the organization is produced.

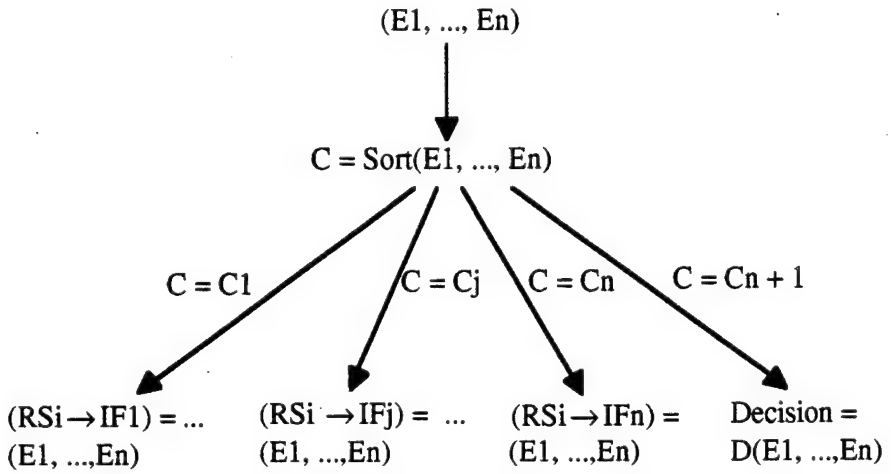


Figure 2.21 Generic Algorithm for the RS Stage

The expression for workload induced by the RS stage of the FDMU dealing with the evidence item Ei, W_{RSi}, is therefore:

$$W_{RSi} = H(E_1,..., E_n) + H(C) + \sum_j H(RS_i \rightarrow IFj) + H(Decision|(E_1,..., E_n))$$

Let us now derive the expressions for the workload for the example described in this chapter:

SA-R:

The variable R takes a value R1, R2, or R3. If R = R1, R1 is sent to the FDMU in charge of the evidence item P; if R = R2, the decision of the organization, En is derived; if R = R3 then R3 is sent to the FDMU in charge of evidence item Q. We have:

$H(SA\text{-}R \rightarrow IF\text{-}Q) = 0$
$H(SA\text{-}R \rightarrow IF\text{-}P) = 0$
$H(Decision \mid R2) = 0$
$H(C) = H(R)$

Therefore, we have:

$$W_{SA\text{-}R} = H(R) + H(C) = 2\,H(R)$$

SA-P:

We have the simple expression:

$$W_{SA\text{-}P} = 2\,H(P)$$

SA-Q:

We have the simple expression:

$$W_{SA\text{-}Q} = 2\,H(Q)$$

IF-P:

Since the FDMU in charge of evidence item P receives only R1 from other FDMU, we have:

$$H(R, Q) = H(R1, Q_{unk}) = 0$$

and therefore:

$$W_{IF\text{-}P} = H(P) + H(R1, P, Q_{unk})$$

IF-Q:

The FDMU in charge of evidence item Q receives information from the other FDMUs. Therefore, we have:

$$W_{IF\text{-}Q} = H(R, P) + H(Q) + H(R, P, Q)$$

RS-P:

This function has for input (R1, P, Qunk). If P = P1 the information (R1, P1) is sent to the FDMU in charge of evidence item Q; if P = P2 then the organization's decision is derived: Friend. We have therefore:

$$H(C) = H(R1, P, Qunk)$$
$$H(RS\text{-}P \rightarrow IF\text{-}Q) = 0$$
$$H(Decision \mid (R1, P1, QU)) = 0$$

and therefore:

$$W_{RS\text{-}P} = H(R1, P, Qunk) + H(C) = 2\,H(R1, P, Qunk)$$

RS-Q:

The FDMU in charge of evidence item Q does not communicate its information to other FDMUs. We have therefore:

$$W_{RS\text{-}Q} = H(R, P, Q) + H(C) + H(Decision \mid (R, P, Q))$$

The distributions of the variables of the algorithms executed inside the functions are computed as follows. For each variable, the possible values that this variable can take are identified and a real number initialized to 0 is associated with each value. A scenario of inputs having a certain probability distribution is defined and used to simulate the set of functions. Each time an input, with probability of occurrence p, is processed, some of the variables inside the functions are used and instantiated to a value, some others are not. The real number associated with the value taken by each instantiated variable is then increased by p. When all the inputs have been processed, the probability distribution of each variable can be deduced by normalizing the real value associated with each value the variable can take. If the variable X can take a value x_1, x_2, ..., or x_n and $r(x_i)$ is the real value associated with the value x_i of variable X, we have:

$$p(X = x_i) = \frac{r(x_i)}{\sum_{j=1}^{n} r(x_j)}$$

The computation of the distribution of the values taken by the variables of the algorithm executed inside the different functions can be done within the Colored Petri Net formalism. This is done by instrumenting the Colored Petri Net representation of the Functional Architecture and requires the definition of additional color sets and functions for the computation of entropy as well as the use of hierarchical pages for a more detailed implementation of these algorithms. Figure 2.22 displays the modified global declaration node containing the pertinent functions and color sets.

```
definetime int start 0;
color Trigger = with trig timed;
color R = with R1 | R2 | R3  | RU ;
var r : R;
color P = with P1 | P2 | PU ;
var p : P;
color Q = with Q1 | Q2 | Q3 | QU;
var q : Q;
color DataInp = product R*P*Q;
color inpid = int;
var i : inpid;
color prob = real with 0.0 .. 1.0;
var pr0, pr1, pr2, pr3:prob;
color Evid = product inpid*DataInp*prob;
color Decis = with Fr | En | Ne | U;
var d : Decis;
color dist = product DataInp*prob;
color choice = with c1 | c2 | c3 | c4 | c5 | c6 | c7 | c8 | c9 | c10 ;
var c: choice;
color cdist = product choice*prob;
color decdist = product Decis*prob;
color problist = list prob;
var pl : problist;
color Entropy = real;
val ln2 = ln 2.0;
fun plogp 0.0 = 0.0
  | plogp(p:real) = ~p * ln(p) / ln2;
fun addlist nil = 0.0
  | addlist ((p::pls)) = p+addlist( pls);
fun divlist (s:prob) (l:prob) = l/s;
fun normlist pl:problist =
   let val su = addlist(pl)
   in if su=0.0 then pl
      else map (divlist(su)) pl
   end;
fun H(pl:problist) = addlist(map plogp pl);
```

Figure 2.22 Modified Global Declaration Node for the Computation of Entropy of Variables

The color set "Evid" has been modified to take into account the probability distribution of the inputs and to distinguish between them. It is a product "inpid" (ID number of the input), "DataInp"

(the value of the different evidence item R, P and Q) and "prob" (the probability of occurrence of the input). The color set "choice" taking values c1, c2, ... c10 is used to represent the internal variables of the algorithms and their values. The color sets "dist" and "cdist" are used to store the real values associated with the values taken by the evidence items (as perceived by each FDMU) and internal variables throughout the net. The color set "problist" is a list of reals and represents in a single token the different real values associated with the possible values taken by a variable. The function "normlist" is applied on such a list to compute the probability distribution of the variable. The functions "plogp", and "addlist" are used in the function "H" to compute the entropy of a variable using formula (1).

Figure 2.23 shows how the instrumentation has been implemented for the function SA-R. The transition SA-R in Figure 2.12 is now a substitution transition, whose decomposition is displayed in Figure 2.23.

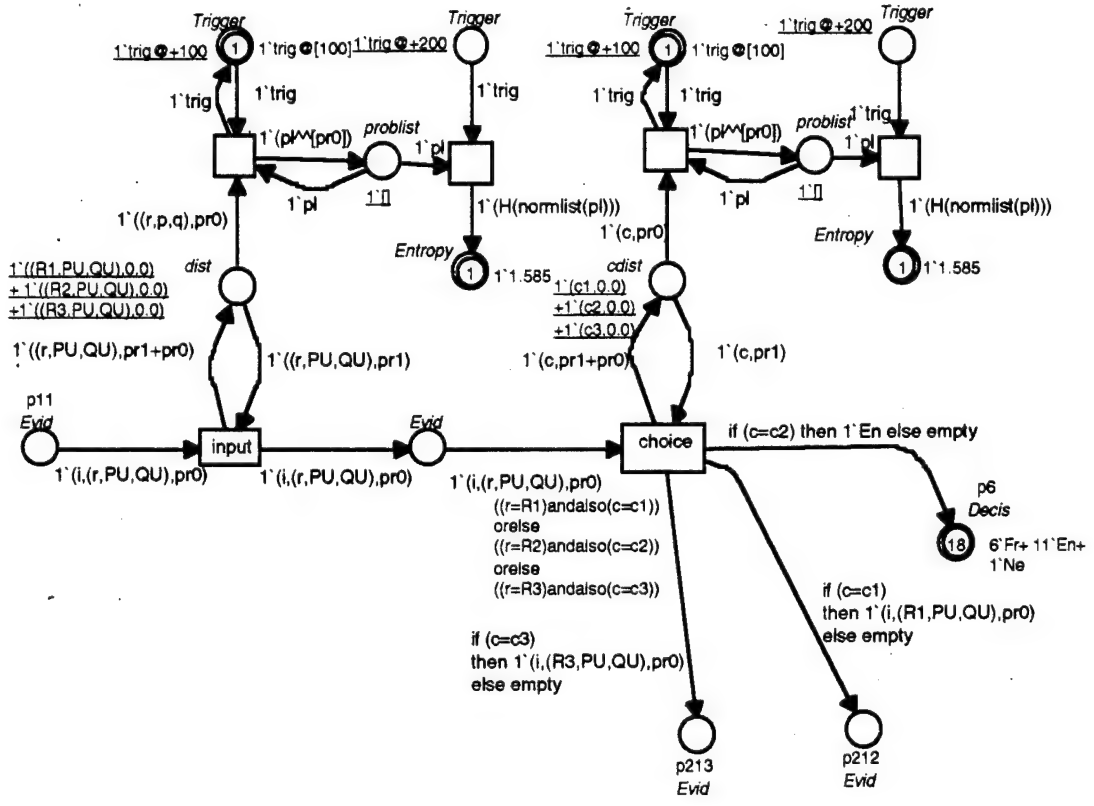


Figure 2.23 Decomposition of Function SA-R

The function SA-R is performed in two steps. The first one is takes into consideration the input as represented by the transition "input". The place with color set "dist", which forms a self-loop with

this transition, is used to store the real value associated with each possible input R. The initial marking of this place is the multi set containing all possible inputs that can be seen by the FDMU with the associated real value initialized to 0. When an input has to be processed, say (R1, PU, QU) with a probability of occurrence of pr0, the corresponding token ((R1, PU, QU), pr1) (where pr1 is the current real value of (R1, PU, QU)) is removed from the place dist and is replaced by the token ((R1, PU, QU), pr1+pr0) where the real value of (R1, PU, QU) has been updated.

The second step is the selection of a course of action according to the value of the input R and is represented by the transition "choice". The internal variable used in this process is c and it can take the value c1, c2 , or c3. In a similar manner as before, the real values for the possible values of this variables are computed.

The operations necessary for computation of the entropy of the different variables begin at time 100, when all the possible inputs has been processed by the organization. The transitions just above the place with color sets "dist" and "cdist" fire as many times as there are tokens in those places to build lists of reals that represents the distribution of the real values of the values taken by the variable under consideration. At time 200, when the construction of the list is over, the second transition can fire to generate the entropy value by applying the function H(normlist()) on the derived list of reals. The list of reals is first normalized to ensure that the sum equals one. Then, the function H is applied on this normalized list to compute the entropy.

The other functions of the model have similar representation. The model has been executed for a scenario in which the 18 possible inputs have the same probability of occurrence (~0.055). The results are listed below.

SA-R: $H(R) = 1.585$ bits/symbol
$H(C) = 1.585$ bits/symbol
$W_{SA\text{-}R} = H(R) + H(C) = 3.170$ bits/symbol

SA-P: $H(P) = 1.0$ bits/symbol
$W_{SA\text{-}P} = 2\ H(P) = 2.0$ bits/symbol

IF-P: $H(P) = 1.0$ bits/symbol
$H(R1, P, Q_{unk}) = 1.0$ bits/symbol
$W_{IF\text{-}P} = H(P) + H(R1, P, Q_{unk}) = 2.0$ bits/symbol

RS-P: $H(C) = H(R1, P, Qunk) = 1.0$ bits/symbol

$W_{RS\text{-}P} = H(R1, P, Qunk) + H(C) = 2.0$ bits/symbol

SA-Q: $H(Q) = 1.585$ bits/symbol

$W_{SA\text{-}Q} = 2\ H(Q) = 3.170$ bits/symbol

IF-Q: $H(R,P) = 0.918$ bits/symbol

$H(Q) = 1.585$ bits/symbol

$H(R,P,Q) = 2.503$ bits/symbol

$W_{IF\text{-}Q} = H(R,P) + H(Q) + H(R,P,Q) = 5.006$ bits/symbol

RS-Q: $H(R,P,Q) = 2.503$ bits/symbol

$H(C) = 1.891$ bits/symbol

$H(\text{Decision} \mid (R, P, Q)) = 1.352$ bits/symbol

$W_{RS\text{-}Q} = H(R, P, Q) + H(C) + H(\text{Decision} \mid (R, P, Q)) = 5.746$ bits/symbol

If the functions performed by a FDMU are all allocated to a single decision maker, we have:

$W_R = W_{SA\text{-}R} = 3.170$ bits/symbol

$W_P = W_{SA\text{-}P} + W_{IF\text{-}P} + W_{RS\text{-}P} - H(P) - H(R1, P, Q_{unk}) = 4.0$ bits/symbol

$W_Q = W_{SA\text{-}Q} + W_{IF\text{-}Q} + W_{RS\text{-}Q} - H(Q) - H(R, P, Q) = 9.834$ bits/symbol.

If we assume that an input has to be processed every 0.5 second and that the bounded rationality constraint threshold F_0 is equal to 18 bits/symbol/s, we have:

$$W_Q / \tau = W_Q / 0.5 = 19.668 \text{ bits/symbol/s}$$

which is larger than F_0 . The bounded rationality constraint is violated. This means that all the functions performed by the FDMU in charge of the evidence item Q can not all be allocated to a single decision maker.

Other function allocation strategies are:

- SA-Q is allocated to a decision maker and IF-Q and RS-Q to another one,
- SA-Q and IF-Q are allocated to a decision maker and RS-Q to another one.

Such function allocation strategies do not violate the bounded rationality constraint.

A procedure has been presented for instrumenting the Colored Petri Net representation of the Functional Architecture of an organization implementing the decision procedures of a DTree. This instrumentation is used to evaluate the workload induced by the different functions and to check whether a proposed strategy for allocating functions to decision makers violates the bounded rationality constraint. The critical parameter F_0 needs to be evaluated independently through pilot experiments as described in Louvet, et al., (1988).

2.9 CONCLUSION

The methodology described in this chapter addresses the design of teams which have to make decisions under conditions of stress and uncertainty. It is a prescriptive methodology that specifies how to design a team architecture that allows for the application of team decision procedures that are executable by human teams under high stress conditions. From an influence diagram, which is a model of the domain in which the team will act, team decision procedures are derived. These decision procedures are represented by a default tree. Each path from the root to a leaf of the default tree corresponds to a sequence of evidence items that have to be examined to reach a conclusion. For each path, a set of fixed structures that models functionally, using Petri Nets, the processes and interactions that have to take place to reach the conclusion is derived. Then, one member from each set is picked and folded with the others to generate a variable functional architecture represented as a Colored Petri Net. From this functional architecture, different team operational architectures are derived by allocation of tasks to resources. A procedure to guide this allocation process, based on the human workload induced by the functions performed by each decision maker, is presented. The interesting aspect of this procedure is that the workload can be evaluated by expanding the Colored Petri Net model of the Functional Architecture. Finally, the derived Operational Architectures can be simulated to generate measures of performance that are used as the basis for the final selection of the team design.

In the next Chapter, we address the question of how to derive team decision procedures from an influence diagram.

CHAPTER III

DERIVING DECISION PROCEDURES

As stated in Chapter II, the first step in the team design process is the derivation of the team's overall decision procedure. This chapter summarizes the work performed on this effort that directly addresses this issue. Section 3.1 describes a class of algorithms for deriving decision procedures that a human team can be expected to execute in real time under conditions of significant stress and uncertainty. The basic idea is to compile a domain model, represented as an influence diagram, down to a set of simple decision procedures, where the expected performance of the decision procedures can be predicted precisely from the influence diagram. Section 3.2 examines the issue of how complex a team's decision procedures must be before they are "near-optimal". More specifically, the analysis attempts to determine classes of decision problems for which simple, humanly-executable decision procedures exist that are near-optimal from problems where near-optimal decision making requires some form of automated support. Such information is essential for determining appropriate team architectures. Finally, section 3.3 examines psychological issues related to the viability of some types of team decision procedures. We argue that even if a team's decision procedures are computationally simple, some procedures are more likely than other procedures to break down (i.e., be executed incorrectly) when team member are operating under stress condtions. A psychological basis for this position is presented and an experiment testing this hypothesis is reported.

3.1 COMPILING INFLUENCE DIAGRAMS

3.1.1 Introduction

There is a growing recognition that Bayesian decision theory provides a powerful foundation upon which to develop automated and partially automated reasoning systems. Decision theory provides a compelling semantics for inference and action under uncertainty, as well as a framework for evaluating the adequacy of heuristic methods. The use of Bayesian techniques is now common place in a number of research areas, including inference (e.g., Pearl, 1987), planning (e.g., Dean and Wellman, 1991), and learning (e.g., Paass, 1991).

Most applications of Bayesian techniques involve problems that require repeated processing of similar cases. A typical example is medical diagnosis, where the objective is to use symptoms and

test results to select the affliction affecting the patient. In such problems, a *belief network* is constructed that encodes a joint probability distribution over a preselected set of conclusions (affliction), evidence items (symptoms and diagnostic tests), and intermediate hypotheses. If decisions and outcome values are included, the network becomes an *influence diagram.* In an influence diagram all possible decisions, and the utility of each decision conditioned on a subset of the evidence items and hypotheses, are explicitly represented. Constructing a good influence diagram involves a substantial knowledge engineering effort. However, once developed, the influence diagram can be used repeatedly to address decision problems which differ in the pattern of evidence that is observed.

There are many decision domains that involve repeated processing of similar problems in time constrained settings. Many examples are found in the command and control arena (sensor interpretation, object identficiation, IFF, etc.). In principle, such tasks are good candidates for the application of influence diagrams. However because of the time constraints, real time processing of influence diagrams is required; as well as real time understanding and (where appropriate) acceptance by users the results of this processing.

Unfortunately, as shown by Cooper (1990), the computational complexity of computing posterior probabilities in a belief network is NP-Hard. This implies that the computational effort required to select optimal decisions using an influence diagram is at least exponential with the size of the diagram. Consequently, in order to realistically apply Bayesian decision theory to time constrained problems, computationally simpler procedures must be developed that approximate exact Bayesian reasoning. Furthermore, there are many decision tasks that people may not wish to delegate to machines (e.g., decision involving life-or-death outcomes). For such decisions, decision making procedures are needed that are humanly-executable, or at least easy enough to understand so that a decision making could effectively use them in a decision aid (see Lehner and Zirk, 1987, and Lehner, et al., 1991, for discussion of user understanding and decision aid usefulness).
Previous researchers have explored three distinct approaches to approximate, rapid processing of influence diagrams. The first is to use simulation algorithms that generate approximate solutions in polynomial time (e.g., Henrion, 1988). The second (e.g., Dean and Wellman, 1991) is to partition the reasoning problem into a series of incremental reasoning steps, and to estimate the computational burden involved before each step is executed. If there is insufficient time to execute the next step, the current solution is offered as an approximation. Unfortunately, both of these approaches generate decision procedures that are not executable by people, or easy for people to understand.

A third approach is to "compile" an influence diagram into a set of simple decision procedures and to apply those decision procedures at execution time. This approach can be used to generate decision procedures that people can execute. Heckerman, et al., (1989) addresses the problem of compiling influence diagrams by generating decision rules that specify, for each evidence state, the maximum expected utility decision. One of compilation approaches proposed by Heckerman, et al., is to develop a *situation-action tree* in which evidence items are sequentially examined until sufficient evidence is accumulated to warrant a decision. In this report, we develop two procedures for generating situation-action trees, and characterize their optimality and computational complexity properties.

3.1.2 Compilation Algorithms

Figure 3.1 depicts an influence diagram. The root node H contains the system's hypotheses. It and the other circle nodes are chance nodes. Each chance node contains a finite number of states with the conditional probability distribution over these states for each possible combination of states of its parent nodes. For example, the arc from node H to B shows the conditional probability of $P(B = b_i \mid H = h_j)$ for each state h_j in node H and state b_i in node B. Rectangle nodes are decision nodes. These nodes contain a set of possible choices. The double line arcs are information arcs. They indicate which information will be available at the time one of the choices in the decision node must be selected. The chance nodes that "feed" a decision node in this way are referred to as evidence nodes. In Figure 3.1, for instance, the state of chance nodes B, E, and F will be known when the choice in Dec must be selected. The diamond node is the value nodewhere it assigns a utility to each row in the cross product of the propositions/decisions of its parents nodes. In Figure 3.1, the utility of the choice selected in Dec depends only the state of H.

In this report, chance nodes that have information arcs going to a decision node are referred to as *evidence items*. An evidence item is *instantiated* when the value of that evidence item is known. For instance, E = e1 asserts that the *evidence value* for evidence item E is e1. A set of evidence values for all evidence items is referred to as an *evidence state*. For instance, the vector <b1, e2, f3> describes the evidence state where B = b1, E = e2 and F = f3.

Once an influence diagram has been defined, there are a variety of algorithms and software tools for processing the network (Buede, 1992). These algorithms can be used to derive the expected utility of any decision, or the posterior probability of any chance node, conditioned on specific values for any subset of the chance nodes.

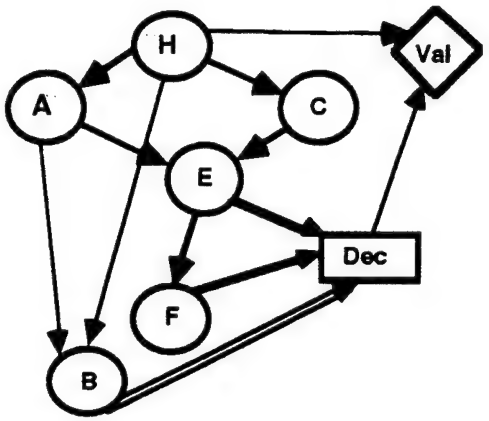


Figure 3.1. An Influence Diagram

Default Trees

We define a *default tree* (DTree) as a tree composed of default nodes (Dnodes) and evidence nodes (Enodes). Each Dnode specifies a decision, while each Enode specifies both an evidence item and a decision. To illustrate, the DTree in Figure 3.2 contains 4 Enodes and 6 Dnodes. This DTree corresponds to a decision procedure where the decision maker begins by either selecting d1 or examining evidence item E. If A is examined and its value is e2, then the decision d1 is immediately selected. If E = e1, then the decision maker selects d2 or examines B. If B = b1, then the decision d1 is selected, else if B = b2 then d2 is selected. Returning to the root Enode, if E = e3 then d3 is selected or F is examined. If F = f1, then d1 is selected. If F = f2 then d3 is selected or B is examined. If B = b1, then d2 is selected, otherwise d3 is selected.

Unless otherwise noted, we will assume that processing of a DTree continues until a Dnode is reached. That is, the evidence item associated with an Enode is always examined and the decision associated with an Enode is never selected.

Dnodes are partitioned into two types. A Dnode is *closed* if the path leading to the Dnode contains all the evidence items available. A Dnode is *open*, if it is not closed. Note that an open Dnode represents a *default decision*, since it specifies decisions that could change if additional evidence is examined.

More formally, we can characterize a DTree as follows. Let ID be an influence diagram which contains decision nodes $\{D_j\}$ and evidence nodes $\{E_i\}$. Let DT be a DTree that contains the nodes

$\{N_i\}$. Each member of $\{N_i\}$ is an open Dnode, a closed Dnode, or an Enode. The following functions are defined with respect to ID.

$path_{DT}(N_i)$ - The set of evidence values in the ancestors of N_i.

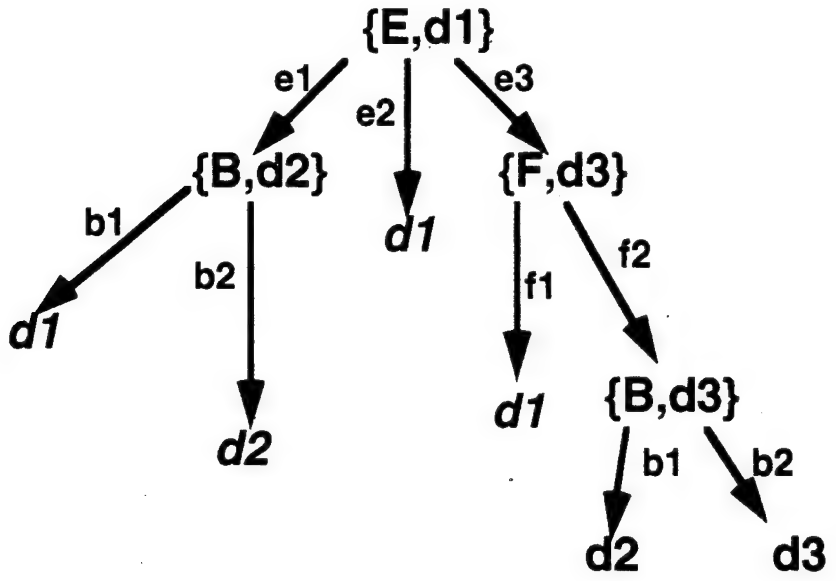


Figure 3.2 A Default Tree (DTree)

For example, if we order the node in Figure 3.2 left-to-right breadth-first, then N_5 is the left-most Dnode and $path_{DT}(N_5) = \{E = e1, B = b1\}$.

evid-path$_{DT}(N_i)$ - The evidence items that are listed in the ancestors of N_i (e.g., *evid-path*$_{DT}(N_5) = \{E,B\}$). For the root node, *evid-path*$_{DT} = \{\}$.

$dec_{DT}(path_{DT}(N_i))$ - The maximum expected utility decisions in ID given the evidence item values leading to N_i. That is, $dec_{DT}(path_{DT}(N_i)) = \max_{dc}[EU(dc \mid path(N_i)]$, where dc is a set that specifies all possible combinations of decisions for the decision nodes in ID (e.g., $dec_{DT}(path_{DT}(N_5)) = \{d1\}$).

Let d be an open Dnode in the DTree DT and E an evidence item. We say that the *expansion* of d with E in DT is the DTree that results from replacing the Dnode d with the Enode {E,d}, and adding Dnodes for each possible value of E. Each new Dnode contains the maximum expected utility decisions. If E_S is a sequence of evidence values, followed by an evidence item, then the *expansion sequence* of d with E_S in DT is the DTree that results by starting at d and sequentially expanding DT with the evidence items in E_S. For instance, the expansion of d with

$E_S = \{E_1 = e1, E_2 = e2, E_3\}$

is obtained by replacing d with the expansion of d with E_1, then replacing the Dnode at $E_1 = e1$ with the expansions of that Dnode with E_2, and then replacing the Dnode at $E_2 = e2$ with the expansion of that node with E_3. An *expansion subtree* is composed of one of more expansion sequences that have the effect of adding a subtree to the DTree. An *expansion set* is composed of multiple expansion subtrees.

$evoi_{DT}(E \mid path(N_i))$ = The increase in expected utility for examining evidence item E given the path to N_i. Formally,

$$evoi_{DT}(E|path(N_i)) =$$
$$(\Sigma_{e \in E} P(E = e \mid path_{DT}(N_i)) \cdot EU[dec_{DT}(\{E = e\} \cup path_{DT}(N_i))])$$
$$- EU[dec_{DT}(path_{DT}(N_i)].$$

$max\text{-}evoi_{DT}(N_i)$ - The evidence item, E, for which $evoi_{DT}(E \mid path_{DT}(N_i))$ is maximal.

$eu\text{-}expand_{DT}(N_i,E_S)$ - The increase in the expected utility of DT that is obtained by replacing $dec_{DT}(path(N_i))$ with the expansion subtree E_S in DT.

It is easily shown that if E_S contains a single evidence node (E), then

$$eu\text{-}expand_{DT}(N_i,E) = P(path_{DT}(N_i)) \cdot evoi_{DT}(E \mid path(N_i)).$$

If E_S is an expansion subtree, then define $mean\text{-}eu\text{-}expand_{DT}(E_S)$ to be the mean value of the individual $eu\text{-}expand_{DT}$ values for the evidence items in E_S.

Definition 3.1 DT-compile: A DTree (DT) DT-compiles an influence diagram (ID) iff every evidence state in ID will lead to a Dnode in DT.
The DTree in Figure 3.2 DT-compiles the influence diagram in Figure 3.1.

Theorem 3.1 DTree Expected Utility: If a DTree (DT) DT-compiles an influence diagram ID, then the expected utility of the DT, with respect to ID, is

$$EU(dec_{DT}()) + \Sigma_{N \in Enodes(DT)} eu\text{-}expand_{DT}(N,e),$$

where e is always set to the evidence item at N.

Proof: (By induction)

Let $\{N_i\}$ be a list of the Enodes in DT ordered in a manner that is consistent with partial ordering induced by the arcs in DT. Let $\{N_j\}_m$ be a subset of the first m Enodes in $\{N_i\}$. Each $\{N_j\}_m$ corresponds to a DTree. By definition,

$$EU(\{N_j\}_1) = \textit{eu-expand}(N_1) + EU(\textit{dec}()).$$

Assume that $EU(\{N_j\}_m) = \sum_{N \in \{N_j\}_m} \textit{eu-expand}_{DT}(N)$.

Again by the definition of *eu-expand*

$$EU(\{N_j\}_{m+1}) = EU(\{N_j\}_m) + \textit{eu-expand}_{DT}(N_{m+1}).$$

Deriving DTrees

The following algorithm can be used to derive a sequence of increasingly complex DTrees.

Algorithm DD

I. Let N_1 be a Dnode containing only $\textit{dec}_{DT}()$.

II. Iterate through the following procedure

A. Select the Open Dnode and evidence item for which $\textit{eu-expand}_{DT}$ is maximal. Call this node N.

B. Set N equal to the Enode $\{\textit{max-evoi}_{DT}(N), \textit{dec}_{DT}(\textit{path}_{DT}(N))\}$

C. For each possible value (e) of $\textit{max-evoi}_{DT}(N)$ add as a subnode to N the Dnode $\textit{dec}_{DT}(\textit{path}_{DT}(N) \cup \{\textit{max-evoi}_{DT}(N) = e\})$.

D. Check stopping criterion. If Stop, then exist with current DTree.

E. Go to A.

In words, DD iteratively replaces a default decision with the evidence item that maximizes the increase in the expected utility of the DTree, and adds as new subnodes the Dnodes that correspond to the best decisions for each possible value of that evidence item.[2] This algorithm is consistent with the situation-action tree development algorithm described informally in Heckerman, et al.,, (1989).

---

[2] Obviously, the efficiency of DD could be increased substantially by recording the results of the *evoi* calculations and keeping track of open and closed Dnodes. However, efficiency improvement will not change the order in which the DTree is expanded. Consequently, they are not presented here.

DD is a greedy algorithm. At each iteration, it expands the DTree by adding and expanding the node that has the greatest increase in the expected utility of the DTree. We refer to such expansions as *greedy expansions*. Although DD is a greedy algorithm, it has two useful optimality properties.

Theorem 3.2 Local optimality of DD: Each Enode added by DD is a locally optimal extension in that if DTx and DTy are both one step expansions of DT, then $EU(DTx) \geq EU(DTy)$.

Proof:

This follows immediately from Theorem 3.1 and the fact that DD always selects the maximum *eu-expand* expansion.

To further characterize the optimality properties of DD, the following definitions are offered.

Definition 3.2 Optimal DTree: DT* is an optimal DTree iff $EU(DT^*) \geq EU(DT)$ for each DTree (DT) where DT has no more Enodes than DT*. Also, DT* is an *optimal expansion* of DT iff DT* is an expansion of DT and DT* is an optimal DTree.

Definition 3.3 E-descending: DT is an E-descending DTree iff for every open Dnode (D), and evidence item (E),

$$P(path(D))evoi_{DT}(E \mid path(D)) \geq P(path(D) \cup \{e_i\})evoi_{DT}(E \mid path(D \cup \{e_i\})),$$

where $\{e_i\}$ is any set of evidence values.

In words, a DTree is E-descending if it is impossible to increase the *eu-expand* value of an evidence item by making it the last element in some expansion sequence.

Theorem 3.3 Optimal Dnode selection: Let DT and DT* be any DTrees where DT is E-descending, DT* is an expansion of DT, and DT* is optimal. If {d} is the set of Dnodes in DT with the maximum *eu-expand* value, then DT* contains an expansion of a node in {d}.

Proof:

P1. Let d be a Dnode in DT and $E_1, ..., E_n$ the evidence items available at d ordered by their *evoi* value. From this ordering, it follows that

$$eu\text{-}expand(E_1 \mid path(d)) \geq eu\text{-}expand(E_i \mid path(d)) \text{ for } i \geq 2.$$

From E-descending it follows that *eu-expand*(E_i | path(d)) ≥
eu-expand(E_i | path(d) ∪ {e_i})) for all {e_i}.

From transitivity it follows that *eu-expand*(E_1 | path(d)) is greater than the *eu-expand* value of an E-node in any subtree rooted at d.

P2. Let d' be a member of {d}. From P1 and transitivity, it follows that *eu-expand*(E_1 | path(d')) is greater or equal to the *eu-expand* value of any E-node in any subtree rooted at an open Dnode in DT.

P3. Let DT* be any expansion of DT that does not contain an expansion of a Dnode in {d}. From P2 it follows that all Enodes in DT', that are not in DT, have an *eu-expand* value that is strictly less than *eu-expand*(E_1 | path(d')). Consequently, a DTree which is the same as DT* except that it replaces a terminal Enode with the *max-evoi* expansion of d' will have a greater expected utility than DT*. Consequently, DT* cannot be an optimal DTree.

P4. The contrapositive of P4 is that if DT* is optimal, then it contains an expansion of a node in {d}.

Theorem 3.4 Optimal Dnode selection by DD: If each DTree (DT) generated by DD is E-descending, then each Dnode selected by algorithm DD for expansion must be expanded in any optimal expansion of DT that contains nodes other than those in DT or {d}, where {d} is the set of Dnodes in DT with the maximum *eu-expand* value.

Proof:

DD selects the Dnode with the maximum *eu-expand* value. Consequently, by Property 3 an expansion of that Dnode must be included in an optimal expansion.

E-descending is not a very stringent constraint. While it is possible to increase the *evoi* value of an evidence item (E) by inserting a path of evidence values ({e_i}) as ancestors to E, a violation of E-descending requires that

$$evoi_{DT}(E \mid path(D \cup \{e_i\})) \; > \; \frac{evoi_{DT}(E \mid path_{DT}(D))}{P(\{e_i\} \mid path_{DT}(D))}$$

That is, the $evoi_{DT}$ value must increase by a multiplier of more than $1/P(\{ei\} \mid path_{DT}(D))$. This can only occur if the increase in the *evoi* value is substantial or if P({e_i}) is near one. Both of these are unlikely if {e_i} involves more than one evidence item. Consequently, violations of E-descending will be infrequent and most violations that do occur will only involve two level expansions.

Theorem 3.4 states that DD always expands the Dnode that must be expanded in an optimal expansion, while Theorem 3.2 asserts that DD always performs a locally optimal expansion of that Dnode. Intuitively, these two properties suggest that E-descending is sufficient to guarantee that DD-generated DTrees are optimal. It isn't. This is because greedy expansions may be redundant given several follow-on expansions. For instance, the *eu-expand* of E_1 may be greater than either E_2 or E_3 individually, but together E2 and E3 may have a greater *eu-expand* than either E1 combined with either E2 or E3. Consequently, an optimal multi-step expansion may include E_2 and E_3, but not E_1.

Unfortunately, the conditions required to guarantee global optimality are very stringent. In effect, it is necessary to assume conditions that imply that for any N, any DTree consisting of N greedy expansions is optimal. It is easy to construct violations of this property where the violations first appear at arbitrary expansion depths. Furthermore, since all DTree expansion procedures eventually lead to the same fully-expanded DTree, all expansion procedures will eventually converge to the same value. As they converge on this common value, there is no reason to believe that the greedy procedures will consistently generate optimal DTrees. Consequently, short of exhaustively searching the space of possible DTrees, there does not seem to be a way to guarantee the generation of optimal DTrees.

On the other hand, the fact that violations of E-descending are not likely to involve the insertion of long evidence chains suggests that DD can be enhanced by examining expansions more than one level deep.

Algorithm DD_n

I. Let N_1 be a Dnode containing only $dec_{DT}()$.

II. Iterate through the following procedure

A. For each Open Dnode find the expansion subtree of depth n or less for which the *mean-eu-expand* value is maximal.

B. Select the Open Dnode for which the *mean-eu-expand* value found in A is maximal. Call this node N and its expansion subtree E_s.

C. Replace N with the subtree that resulted in the maximal *mean-eu-expand* $_{DT}(N)$.

D. Check stopping criterion. If Stop, then exist with current DTree.

E. Go to A.

DD_1 is the same as DD. DD_n is similar to DD, except that it will look n levels deep to find the expansion with the greatest average contribution to the expected utility of the DTree. We call such expansions *greedy n-step expansions*. Since DD_n examines strictly more nodes than DD, it will usually generate DTrees with expected utility greater or equal to the DTrees generated by DD. However, since neither algorithm is globally optimal, this cannot be guaranteed.

DD_n satisfies local optimality under weaker conditions than DD.

Definition 3.4 E_n-descending: DT is an E_n-descending DTree iff for every open Dnode (D), and evidence item (E),
$P(path_{DT}(D))\cdot evoi_{DT}(E \mid path(D)) \geq P(path_{DT}(D) \cup \{e_i\})\cdot evoi_{DT}(E \mid path_{DT}(D \cup \{e_i\}))$,
where $\{e_i\}$ is any set of evidence values with cardinality not less than n.

Note, E-descending is equivalent to E_1-descending.

Theorem 3.5 Local optimality of DD_n: If each DTree generated by DD_n is E_n-descending, then the *mean-eu-expand* value of each expansion subtree selected by DD_n is greater than or equal to the *mean-eu-expand* value of any alternative expansion set.

Proof:

As shorthand, let *meu* = *mean-eu-expand.*

P1. We first show that for any DTree, there is a maximal *meu* expansion set which is a subtree. Let M be a maximum *meu* expansion set of DT. If M is not a single subtree, then M must be composed of a set of subtrees $\{M_1, ..., M_k\}$, each of which has its root at an open Dnode in DT. Select a subtree M_i in $\{M_1, ..., M_k\}$ for which $meu(M_i) \geq \max[meu(M_1), ..., meu(M_k)]$. From basic algebra it follows that $meu(M_i) \geq meu(M)$.

P2. Next we show that for any open Dnode there is an expansion subtree with depth no greater than n for which *meu* is maximal. (The root node of an expansion subtree is at depth 1.) Let M be a maximal *meu* expansion subtree that contains at least one node with depth greater than n. Let $\{m_1, m_2, ..., m_i, m_{i+1}, ..., m_k\}$ be the Enodes in M, where m_1 is the root, m_2, .., m_i are all the nodes of depth n or less. Since M is a maximal *meu* expansion subtree, it follows that $meu(M) = meu[\{m_1, m_2, ..., m_i, m_{i+1},..., m_k\}] \geq meu[\{m_1, m_2, ..., m_i\}]$; otherwise $m_{i+1}, ..., m_k$ would not be included in M. From basic algebra it follows that $meu[\{m_{i+1}, ..., m_k\}] \geq meu[M]$. Let E* be the evidence item with the maximal *eu-expand* value. From E_n-descending it follows the *eu-expand*(E*) is greater

than the *eu-expand* value of any possible node at depth n or greater. Therefore, *eu-expand*(E*) $\geq$ max(*eu-expand*((m_{i+1}), ..., *eu-expand*(m_k)), which implies *meu*(E*) $\geq$ *meu*(m_{i+1}, ..., m_k). Therefore, *meu*(E*) $\geq$ *meu*(M).

Consequently, the expansion subtree {E*} is also a maximum value expansion subtree.

P3. DD_n always selects the expansion subtree with depth $\leq$ n with a greatest *meu* value. Therefore, it follows from P1 and P2 that DD_n always selects an expansion set for which *meu* is maximal.

Algorithm DD_n allows the influence diagram compilation process to be arbitrarily conservative. Indeed, if n is set to the number of evidence items, then DD_n will exhaustively search the space of all DTrees. However, as noted above, violations of E-descending that are greater than two levels deep are very unlikely. Consequently, algorithm DD_3 will almost certainly generate a sequence of expansions that are locally optimal for *any* search depth. DD_n also satisfies the optimal Dnode selection property described in Theorem 3.4. This is because, whenever a DTree is E-descending, DD_n will select the same expansion as DD.

Computational Complexity

Let CI be the average computational complexity of processing an influence diagram. Let NE be the average number of evidence values for each evidence item. With each iteration of algorithm DD_n, the number of times the influence diagram is processed in NE^n. If a DTree contains R nodes, then the computational complexity of generating that DTree was $NE^n(R)(CI)$. That is, if the size of the DTree if fixed *a priori*, the computational burden of generating a DTree using DD_n is a linear function of the computational burden of processing the influence diagram.

3.1.3 Applications

Performance evaluation presumes a model of the decision situations that a rule set is designed to handle, along with assessments of the probabilities and utilities associated with those situations. Otherwise, it would be possible to make a rule set look arbitrarily good or bad by carefully selecting the decision situations the rule set is tested against

Each form of evaluation can provide a guide to the process of *generating* rule sets. For instance, Remy and Levis (1988) and Zaidi (1991) use concepts of architectural acceptability to derive a space of candidate architectures. These architectures, in turn, limit the types of rule sets that can be generated. The principal result of this section is that a performance evaluation model (i.e., an

influence diagram) can be used to derive procedure rules. In particular, the probability/utility information that is needed to evaluate a rule set is "compiled" into a DTree which defines a rule set that is logically complete, consistent, humanly-executable, and near-optimal in expected utility.

Note also, that the DTree formulation supports adaptation to temporal and workload constraints, as shown in Chapter II. Recall that there is a default decision associated with each node in a DTree. As a result, processing of a DTree can be terminated at anytime with a decision. This behavior can be represented within the DTree formulation by inserting additional Enodes, where time/workload information is the evidence that is examined. If there are severe time/workload constraints, then the Enode branches to a Dnode with a default decision. If processing time is available, then the Enode branches to the next evidence to consider.

Finally, we note that the overall objective of this research is to develop near-optimal decision procedures that can be quickly and reliably executed by a team of human decision makers. The specification of a DTree is the first step in the process of specifying a team's decision procedures. The DTree must still be partitioned into several decision procedures that can be allocated to different team members, as shown in Chapter II.

3.2 NEAR-OPTIMAL DECISION PROCEDURES

3.2.1 Introduction

In C^2 there are numerous situations where important decisions must be made under stress conditions based on limited data. In such situations, it is essential that a C^2 team have a good set of operating procedures. These procedures should be clear and simple enough to be executable in high stress conditions, yet sufficiently reliable to guarantee that appropriate decision are made given the time and data available. Unfortunately, theoretical results in decision theory (e.g., Cooper, 1990) suggest that even for decision problems where the number of decision options and evidence states are finite, the complexity of optimal decision making is at least NP-Hard. This suggests that "simple" and "good" are involved in a severe trade-off.

In section 3.1 we presented a methodology for deriving near optimal decision procedures that could be executed by human teams. This methodology uses influence diagrams to model the decision situations that a team may face. The influence diagram is then compiled into a set of decision rules. The procedure is iterative, with each iteration additional rules are added to the

decision procedure. With each iteration, the expected utility of the decision procedure increases and will eventually asymptote at the expected utility of an optimal decision procedure.

This section examines how complex a decision procedure must be before it is near optimal. Although the complexity of optimal decision making increases exponentially with the size of the influence diagram (with respect to the number of arcs and propositional variables in the diagram), there may be classes of decision problems where a simple decision procedure (e.g., a small rule set) will generate optimal or near optimal performance. The objective of the study describe below was to identify such classes of decision problems.

3.2.2 Method

The experimental method was to generate influence diagrams randomly, calculate the expected utility of an optimal decision procedure for that influence diagram, generate a series of increasingly complex DTrees, and compare the expected utility of the DTrees with the optimal decision procedures. Each of these steps is described below.

Generate Influence Diagrams

Influence diagrams were generated that varied in accordance with 2x2x2x2 factorial design. The following factors were used.

In-degree level. The in-degree of a node is the number of parents of that note. The indegree level of the chance nodes was set at either one or three. Figure 3.3 depicts diagrams with indegree limits of one and three respectively. Note that since a diagram must be acyclic, there will be some nodes with fewer parents than the indegree level.

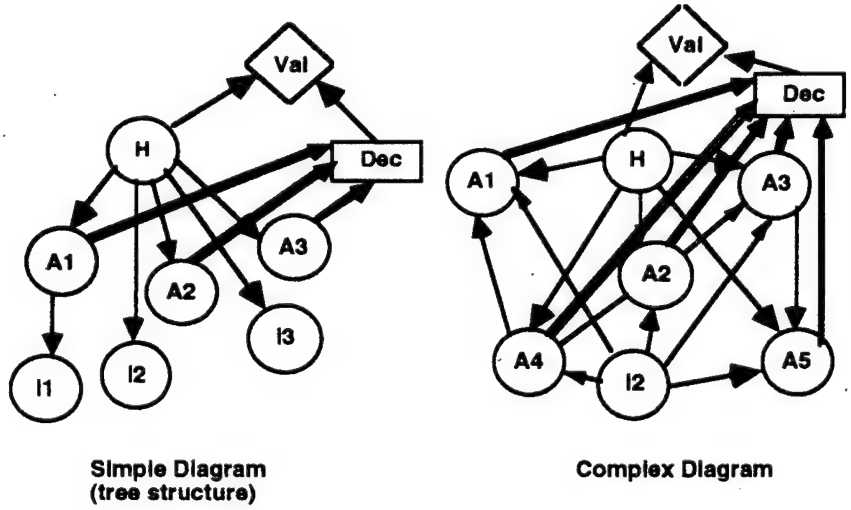


Figure 3.3 Two Examples of Networks Generated During the Monte Carlo Study

Number of hypotheses. In all the influence diagrams generated for this study the choices in the decision node corresponded one-to-one with the states in the root chance node, which we refer to as the hypothesis node. The number of states in the hypothesis node was set at either two or four. Note that in all the diagrams generated, the hypothesis node was the root chance node.

Number of evidence nodes. The number of evidence nodes was set at either three or five. Recall that an evidence node is defined as a chance node for which the state of that node is known when the decision must be made.

Number of evidence states. The number of states associated with each evidence node was set at either two or four.

Five influence diagrams were generated randomly for each cell in the 2x2x2x2 design matrix. Each diagram had one decision, one value, and seven chance nodes. The value node assigned a utility of 1.0 if a selected decision corresponded to the true state of the hypothesis node (H), and a utility of 0.0 otherwise. Consequently, the expected utility of a decision procedure was equal to the probability that the decision procedure would select the true hypothesis state. Each of the conditional probability values in the diagram was assigned randomly. All random assignments in generating the diagrams were from a uniform distribution or density function.

Derive DTrees

A sequence of increasingly complex DTrees were derived using algorithm DD. Briefly summarized, this algorithm works as follows.

Algorithm DD for experiment

i. Determine the decision with the highest expected utility when there is no evidence available. A single Dnode containing this decision is the initial DTree.
ii. For each Dnode, select the evidence item has the greatest increase in the expected value of information. For each Dnode, the evidence item with the greatest expected value of information will be the next possible expansion of the Dnode.
iii. Select the Dnode which has the next possible expansion with the greatest increase in the overall expected utility of the DTree. Replace this Dnode with an Enode (containing the selected evidence item) and add a Dnode for each possible evidence state for the Enode.
iv. Continue looking for more evidence data (iterate on ii and iii) until there is no evidence is left or a threshold on the increase in expected utility is reached.

Calculate expected utility of optimal decision procedure

A complete evidence state is defined as a vector that specifies the state of each of the evidence nodes. For any evidence state, the maximum expected utility choice can be derived using any of a variety of algorithms for processing influence diagrams. Furthermore, the probability of an evidence state can be derived using any of a number of algorithms for processing belief networks. Consequently, the expected utility of an optimal decision procedure is equal to sum over the probability of each evidence state times the expected utility of the optimal choice within each state.

Dependent Variable

The principal dependent variable in the study was relative expected utility, which is defined as the expected utility of the DTree divided by the expected utility of the optimal decision procedure.

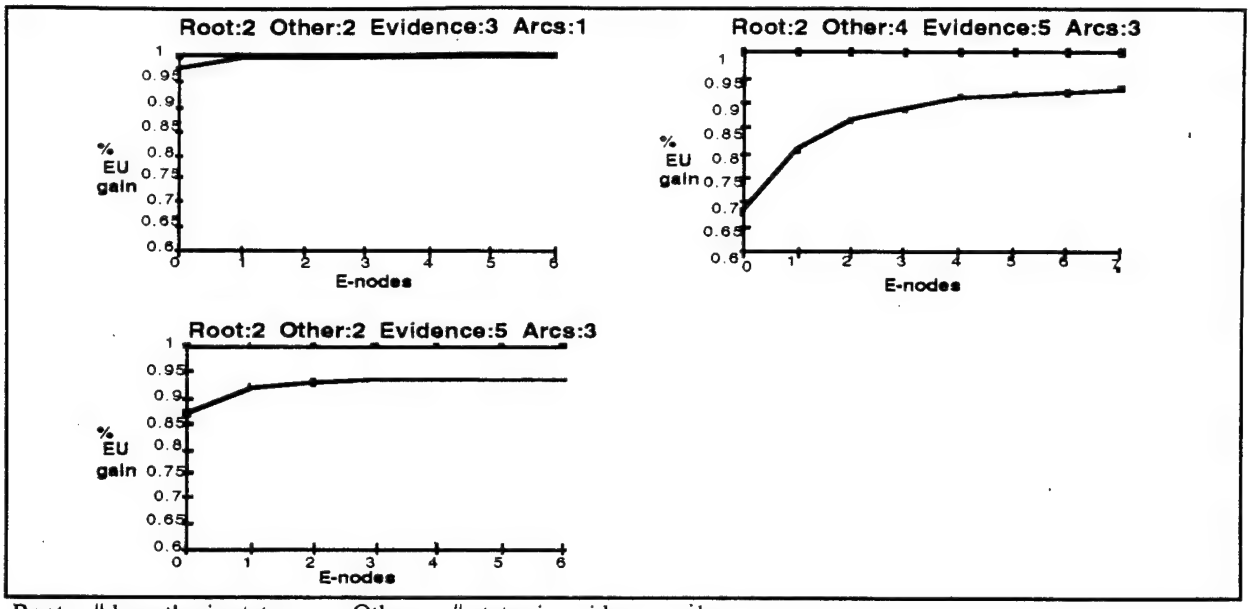
3.2.3 Summary of Results

A complete report of the results will be found in Sadigh (1993). Here we summarize the main findings.

For each cell in the factorial design, the results were summarized by graphing the increase in DTree relative expected utility as E-nodes were added to the DTree. In general, we observed that the combinations of increase in the indegree level and multiple propositional variables, in both the hypothesis and evidence nodes, were the influential factors in increasing the complexity of the decision procedures. The increase in number of evidence nodes in the influence diagram had a relatively small effect.

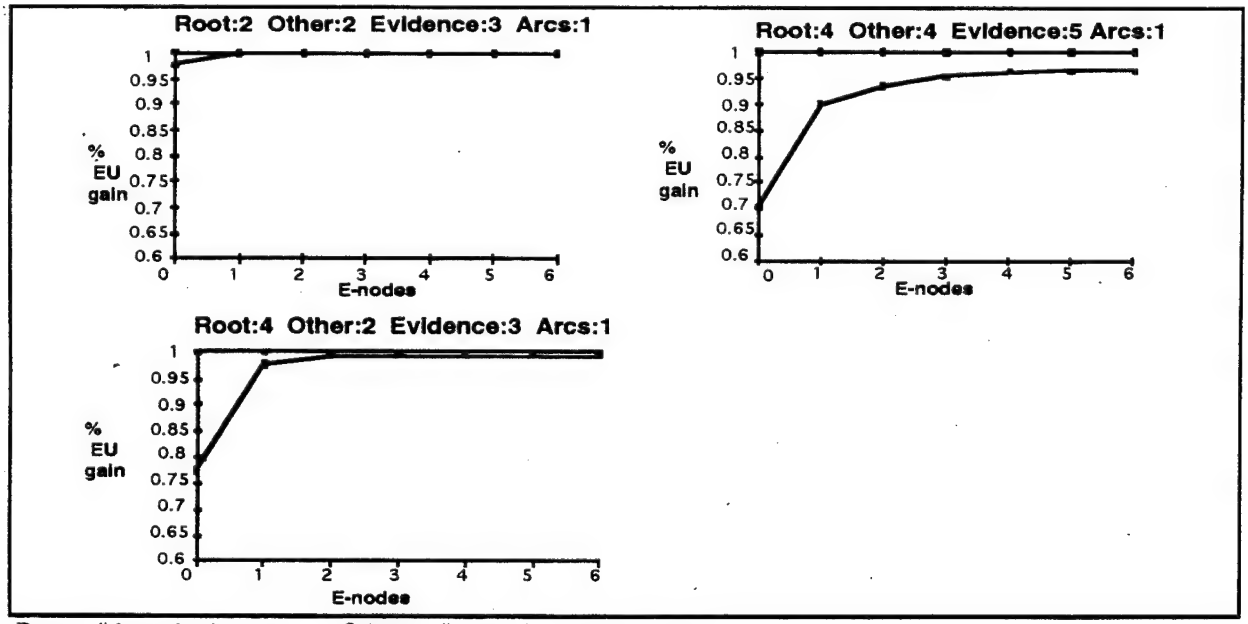
When there were only two hypothesis states, simple DTrees (four or fewer E-nodes) with relative expected utility greater than 0.9 could almost always be found. Figure 3.4 shows three graphs illustrating this result.
The results in Figure 3.5 depicts another situation where simple decision procedures could generally be found. In this case, when the structure of the influence diagram was a tree, DTrees with relative expected utility greater than 0.9 could usually be found.

On the other hand, Figure 3.6 shows the results for some situations where simple DTrees with high relative expected utility could not usually be found. In the case where there are four states for the hypothesis and evidence nodes, and an indegree level of three, relative expected utility leveled-off at around 0.8.



Root = # hypothesis states Others = # states in evidence nodes
Evidence = # evidence nodes Arcs = Indegree level

Figure 3.4 Sample of Results with Number of Hypothesis States = 2



Root = # hypothesis states Others = # states in evidence nodes
Evidence = # evidence nodes Arcs = Indegree level

Figure 3.5 Sample of Results with Number of Indegree Level = 1

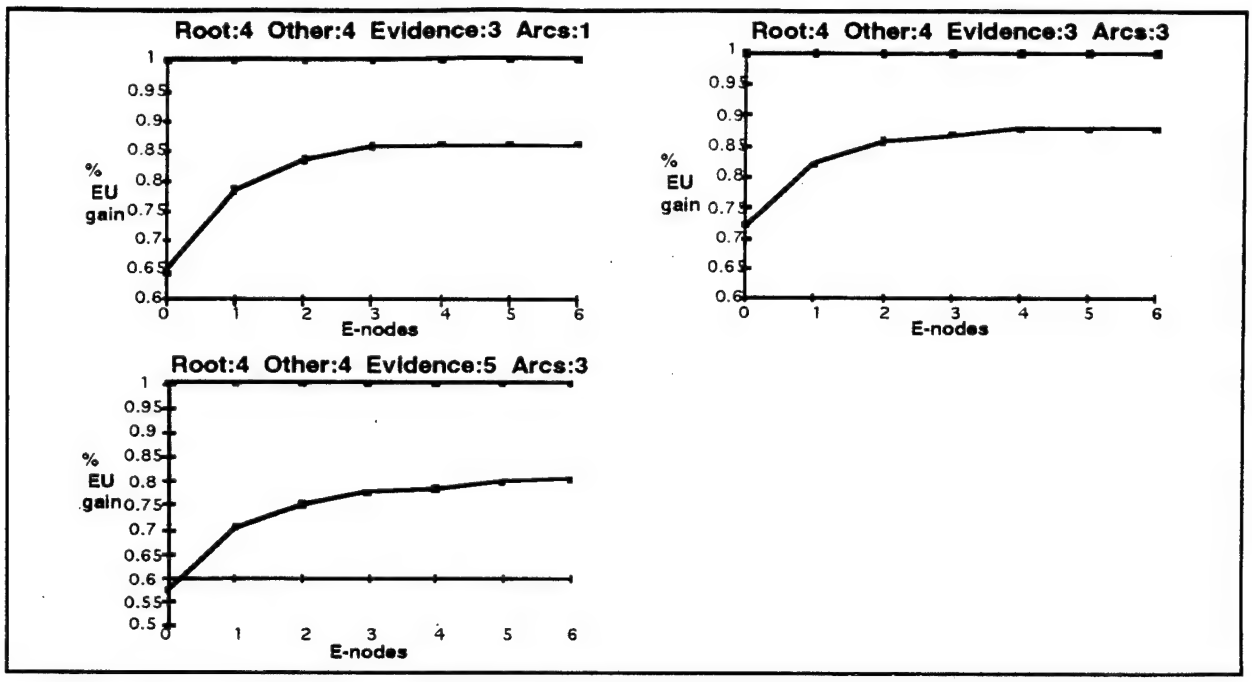


Figure 3.6 Sample of Results with Multiple Hypotheses and Indegree Level = 3

3.2.4 Discussion

The results suggest that simple decision procedures (e.g. standard operating procedures - SOPs) are often adequate for addressing decision problems where the decisions are binary (e.g. identifying friends and foes - IFF), or when there are multiple independent sources of information, (i.e., a tree structured influence diagram) In these circumstances, it is usually possible to derive simple decision procedures with EU which is comparable to that of an optimal decision procedure. For problems of this type, it seems likely that a carefully thought out set of operating procedures will exhibit optimal or near-optimal performance. There is probably little benefit to be gained from developing an extensive domain model (such as an influence diagram, analytic model, or computerized simulation) that could be used to derive/test alternative decision procedures. Also, it would seem that such problems are not good candidates for decision support.

On the other hand, when there are there are multiple choices, and the decision problem involves multiple interacting sources of information, then it is unlikely that a simple decision procedure exists which will generate near optimal performance. In such circumstances, there may be considerable benefit to be gained from an extensive effort to formally model a problem. Such models may be needed to effectively assess the performance of alternative decision procedures.

Furthermore, because simple decision procedures may not suffice, such problems seem to be good candidates for decision support.

3.3 COGNITIVE BIASES IN TEAM DECISION PROCEDURES

3.3.1 Introduction

In section 3.2 an approach to deriving a team decision procedures was described. This approach emphasized the use of analytic procedures to derive team procedures that exhibit both high performance and low workload. It was implicitly assumed that alternative procedures with equal workload would be executed with equal reliability. In this section, we argue that this is an oversimplification. Instead, we argue for the general hypothesis that "unnatural" decision procedures are not likely to be reliably executed under conditions of time stress. The experiment reported below is a test of this hypothesis.

Team Decision Making

A *team* is defined as a group of decision makers, with overlapping areas of expertise, that work cooperatively to solve common decision problems. The cockpit crew of a commercial aircraft (pilot, co-pilot, navigator) is an example of small team, while the crew of a submarine control room is an example of a somewhat larger team. Command and control teams (C^2 team) are further characterized as teams where (a) each member of the team is responsible for an assigned set of tasks, although the assignment of tasks may change dynamically, (b) each team member has been trained for the tasks that he or she is to perform, and (c) for the problems the team faces, the team members have the common goal of satisfactorily solving that problem. Finally, we note that C^2 teams often address problems that are severely time constrained.

C^2 teams are generally well-trained. Consequently, C^2 team performance is usually high. However, "human error" does sometimes occur. This is particularly true under conditions of stress. If the consequences of decisions are severe, and there is very little time for the team to perform its tasks, catastrophic mistakes can easily occur. The objective of this research is to experimentally identify team decision procedures that tend to be executed relatively unreliably under conditions of time stress.

Cognitive Biases and Behavioral Decision Theory

Behavioral Decision Theory (BDT) is a branch of psychology that performs research in human judgment and decision making (JDM). Although related to cognitive psychology research in such

areas as human problem solving (e.g., Newell and Simon, 1972) memory, natural language understanding, and the like, it has traditionally been separated from these areas by two features (Lehner and Adelman, 1990). First, there is an emphasis on tasks that involve quantitative tradeoffs and the integration of subjective judgments (e.g., probability assessment, option selection). Second, there is more of a focus on models that characterize JDM performance rather than JDM processes. For instance, BDT researchers often use linear models to characterize behavior in contexts where it is believed that the "internal" JDM process is pattern-based and non-linear.

An important result in the BDT literature is that human decision making behavior exhibits a number of *cognitive biases*. Cognitive biases are defined as judgments that consistently deviate from a normative ideal. The standard explanation for the occurrence of cognitive biases is that people employ a variety of heuristic procedures when making judgments that bear little resemblance to theoretically normative procedures. Some of these biases are listed below.

Availability Bias. People often overestimate the probability of an event that is easy to recall or imagine (e.g., Tversky and Kahnemann, 1973).

Confirmation Bias. People tend to seek and focus on confirming evidence, with the result that once they've formed a judgment, they tend to ignore or devalue disconfirming evidence (e.g., Wason, 1960; Tolcott, et al., 1989).

Frequency Bias. People often judge the strength of predictive relations by focusing on the absolute frequency of events rather than their observed relative frequency. As Einhorn and Hogarth (1978) have shown, information on the nonoccurrence of an event is often unavailable and frequently ignored when available.

Concrete Information. Information that is vivid or based on experience or incidents dominates abstract information, such as summaries or statistical base-rates. According to Nisbett and Ross (1980) concrete and vivid information contributes to the imaginability of the information and, in turn, enhances its impact on inference.

Conservatism. If people are forced to consider base rates, then they often underestimate the predictive value of new information. That is, their revised probability estimates remain too close to the original base rates (Edwards, 1968).

Anchoring and Adjustment. A common strategy for making judgments is to anchor on a specific cue or value and then to adjust that value to account for other elements of the

circumstance. Usually the adjustment is insufficient. So once the anchor is set, there is a bias toward that value (Kahneman and Tversky, 1973).

"Law of Small Numbers". Problems can be framed in such a way that people, including trained statisticians, give undue confidence to conclusions supported by a relatively small amount of data (Tversky and Kahneman, 1971).

Hindsight Bias. This is perhaps the most problematic of all biases (Fischhoff, 1975). After an event occurs, people will often claim they predicted the event ("I knew it all along!"), even though prior to the event they were very uncertain.

Fundamental Attributi on Error. People tend to attribute success to their own skill and failure to chance or the circumstances in which they were situated. However, when evaluating the performance of other people, the tendency is to attribute other people's failure to their personality traits, not the situation (see Nisbett and Ross, 1980).

Because of the prevalence of cognitive biases, there has also emerged a research literature addressing the "debiasing" problem. The objective of this research is to develop presentation and problem solving techniques that reduce biases. This literature is reviewed in O'Connor (1991). Despite a few successes, the principal result seems to be cognitive biases are very resistant to debiasing techniques.

Cognitive Biases and Team Decision Making

The principal focus of the cognitive bias literature is on natural decision making procedures. Consequently, it can be argued that cognitive biases are irrelevant to C^2 team decision making, since team members are well-trained in their decision making tasks. This argument asserts that, as long as workload is not excessive, people will reliably execute whatever decision procedures they have been trained to execute; irrespective of what they might naturally do if they were not trained. In contrast, one could argue that "unnatural" decision procedures are *vulnerable-to-bias*. A vulnerable-to-bias procedure is defined as a judgment or decision procedure that an untrained or poorly trained decision maker would not reliably execute, because of a cognitive bias. For instance, a judgment procedure that requires a person to make an initial judgment based on sparse evidence, and then to update that judgment based on additional evidence, is vulnerable to both the conservatism and the confirmation bias, as well as to any other biases that might result from the use of an anchoring and adjustment heuristic. These biases suggest that people would normally

undervalue the impact of new evidence, and that they may stick with their original judgment in the face to strong counter evidence.

Vulnerable-to-bias procedures may impact team decision making in two ways. First, it may be difficult to train decision makers in decision procedures that are vulnerable-to-bias. The debiasing literature (O'Connor, 1991) suggests that this is likely to occur. Second, under conditions of stress, it is possible that vulnerable-to-bias decision procedures will not be as reliably executed as more natural decision procedures. It is this later possibility that is investigated here.
The experiment reported below contrasts two perspectives.

> *Perspective 1 (P1)* - Cognitive biases are largely a matter of preference. Although people tend to use heuristic rules that deviate from normative decision theory, they can be taught to reliably use alternative rules; as long as the alternative rules do not exceed workload constraints.

> *Perspective 2 (P2)* - Cognitive biases are largely a matter of capability. Even after training, people do not reliably execute judgment and decision procedures that are vulnerable-to-bias.

For team decision making under stress, these two perspectives differ considerably with respect to their implications for team design. If P1 is correct, then the literature on human cognitive biases is largely irrelevant to the problem of designing teams. Properly trained and practiced teams will reliably execute correct decision procedures until workload or other bounded rationality constraints are exceeded. On the other hand, if P2 is the correct perspective, then cognitive bias considerations should place severe constraints on the design of a team. Specifically, team architectures and decision procedures should avoid vulnerable-to-bias decision procedures, since these decision procedures will not be reliably executed in high stress conditions.

3.3.2 Method

The experiment was a modification of that of Jin (1990). Two person teams worked together to defend a battle group from incoming aerial attacks. Each team member must assess the type of aircraft associated with each radar track.

Subjects

Subjects were paid volunteers from the graduate and undergraduate student population at George Mason University. Data from eleven two person teams was collected. In addition, pilot data from four two person teams was collected.

Materials

This game was played with two players and three Macintosh LC computers. Each player was placed behind one computer. The players were named DM1 and DM2 on the computer. The third computer was named DM3. In this experiment the DM3 computer was only used to synchronize the game between between DM1 and DM2.

Display

The display was divided into six windows. Figure 3.7 shows the display from DM1's perspective. The display and all procedures for DM2 is a mirror image of DM1.

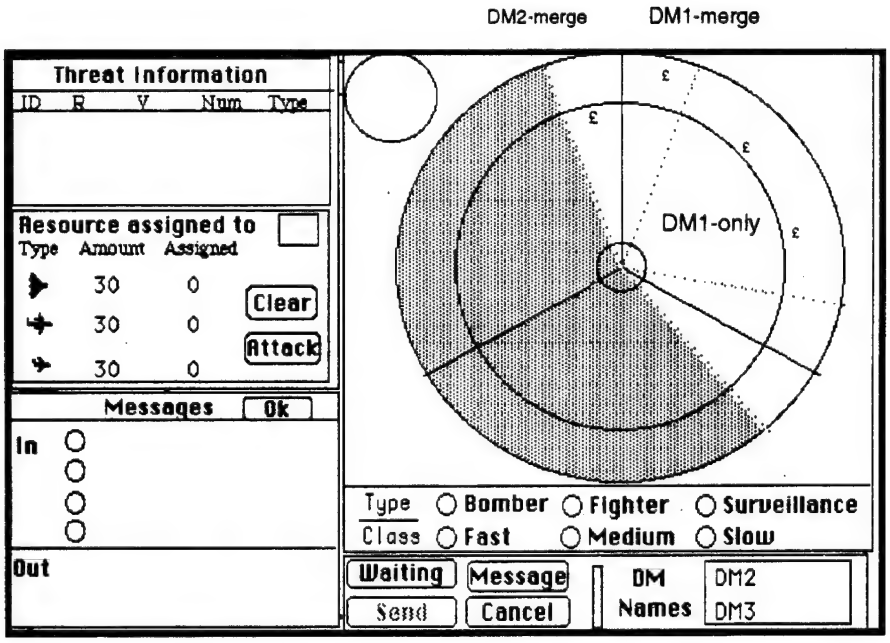


Figure 3.7 Screen Display for DM1

The first window (upper left) was for displaying Threat Information. When a threat was selected, (see below) information about that threat was displayed. This information includes an ID number,

the angle of attack on the screen (R), the velocity of the track (V), and the number of aircraft in the track (Num). In addition, if the other DM sent advice regarding the type of threat (see below) that information was also displayed (Type). However, this Type advice was not displayed until after the DM made an initial judgment as to the type of the aircraft.

The second window (below the first window) displays the resources assigned to each threat after the threat and the Attack button were selected. This region includes two buttons, Attack and Clear. The Clear button was not used in this game. The Attack button was routinely selected after target Type was selected.

The third window, which is located below the second region, is the Messages window. Messages from the other player were displayed in this region. In this experiment, the only information that was displayed was the other DMs assessment as to the type of aircraft in a track. This Type "advice" was only displayed after an initial Type judgment was made by DM1.

The fourth window is the Radar. The Radar display was located on the top right hand side of the screen. In this region the incoming threats were displayed by a small triangle and the players selected individual threats by placing the cursor on the threat and clicking the mouse button. Once this action is carried out, information about the threat was displayed in the Threat Information window. From DM1's perspective radar, the radar was partitioned into four regions. They are DM1-merge, DM2-merge, and DM1-only and the region where no information was displayed. For the DM1-only region, DM1 could click on a track; make judgments with respect to its Class and Type; and hit the Attack button. For the DM2-merge region, DM1 could click on a track; make judgments with respect to its Class and Type; and then send that advice to DM2 by pressing the Send button. For the DM1-merge region, DM1 could click on a track; make judgments with respect to its Class and Type; review the advice from the other DM; change his or her Type judgment; and click the Attack button.

In the remainder of this section, the regions on the Radar screen will be generically referred to as DMi-only, DMi-merge, DMj-merge.

The fifth window is located below the radar screen. It is used to record Class (Slow, Medium, Fast), and Type (Bomber, Fighter, Surveillance) judgment for the threats that appear on the Radar. These are selected based on the threat information that appear in the Threat Information window and advice that appears in the Messages and Threat Information windows.

The sixth window is located at the bottom right hand side. In this window there are four keys: Wait, Message, Send, and Cancel. This region was used to transmit type advice from one DM to the other. This was carried out by selecting the destination from the list of DMs, selecting the Message button and then selecting the Send button.

Procedures: Objective and Rules of the game

The team objective was to correctly identify and attack all the threats before they reached the small center circle. For all three regions, each DM must initially to use number and velocity information to select Class, Size, and Type. The rules for these initial judgments were as follows.

Class

Slow	if Velocity < 500
Fast	if Velocity > 800
Medium	otherwise

Size

Small	if Number < 5
Medium	if Number = 5
Large	if Number >5

Type.

Fighter	if Speed = Slow and Class = Small
	if Speed = Slow and Class = Medium
	if Speed = Fast and Class = Large
Bomber	if Speed = Slow and Class = Large
	if Speed = Medium and Class = Large
Surveillance	otherwise.

For threats in the DMi-only or DMj-merge regions either the Attack or Send button was hit after the Type judgment was made. For the DMi-merge region, after Type was selected, DM2's advice was displayed, if any had been sent. At this point DMi was trained to revise his or her Type judgment according to the following rules.

Condition	Initial Type Judgment	Type Advice Received	Final Type Judgment
FF	Fighter	Fighter	Fighter
FB	Fighter	Bomber	Bomber
FS	Fighter	Surveillance	Surveillance
BF	Bomber	Fighter	Bomber
BB	Bomber	Bomber	Bomber
BS	Bomber	Surveillance	Fighter
SF	Surveillance	Fighter	Surveillance
SB	Surveillance	Bomber	Fighter
SS	Surveillance	Surveillance	Surveillance

Note that the rules for the FB and FS condition require the DM to modify his or her original opinion to conform with the new information (the other DM's advice). Also the BS and SB conditions require that the DM select a new Type that that was inconsistent with both DMs' initial selection. The decision procedure for the FB, FS and the BS, SB conditions were considered to be vulnerable-to-bias, since they required subjects to anchor and then adjust their original judgment.

Instructions

Subjects were instructed about the general nature of the game. They were told that the tactical information for threats arriving in the merge (i.e., DM1-merge and DM2-merge) regions was not as reliable as the information in the DMi-only regions. Consequently, the rules for combining initial Type judgment with the other DM's advice were needed to increase accuracy.

Training

The subjects were trained for a period of approximately 1.5 hours before the game started. Training occurred in four stages.

During the first stage, the players were trained in their DMi-only region. They were instructed to ignore tracks arriving in the other regions. This continued until the subjects were comfortable with the rule set. The training set involved tracks at varying speeds (see below).

Once the players were comfortable dealing with the threats in DMi-only region, they were instructed to ignore this region and concentrate only on the threats in the DMj-merge region, where they had to make a Type judgment and send that judgment as Type advice to the other DM. Once

again, they repeated this (at varying speeds) until they were comfortable with the rules and procedure.

Third, the players were instructed to practice sending advice and then to deal with their DMi-merge region. Here they were required to learn the rules for adjusting their initial judgments on the basis of the other DM's advice. Once again this continued until the subjects were comfortable with the rule set and general procedure.

Finally, they were told to practice the complete game and try to stop all threats before they came to the center of the radar. It was recommended that subjects process tracks in the DMj-merge, DMi-only, and the DMi-merge regions respectively. This guaranteed that the other DM had time to send advice before the subject began processing his or her DMi-merge tracks.

This practice session was repeated until the players felt proficient with the rules.
In all of these practice sessions the players were allowed to look at two tables that summarized the judgment rules. This was done in order to let them confirm that their decisions were correct.

Experimental Tasks

Each team processed 12 sets of ten trials each. The 12 sets were partitioned into four sets each with tracks moving at 400, 600, 750, and 900 mph respectively. At a speed of 400 mph the subjects had 45 seconds to process all tracks. At 600 mph, they had 30 seconds. At 750 and 900 mph, they had 24 and 20 seconds respectively. There was a short break between trials (approximately 20 seconds) and a longer break between sets of trials (about four minutes). The training and experimental sessions were distributed over two days. Each team took a total of about 2.5 hours to complete all 12 sets of trials. Consequently, teams spent an average of about 4 hours in training and experimental sessions.

It should be noted that the Appletalk network crashed about once every 6 hours during training and experimentation. Consequently, it was sometimes necessary to have a short break during a set of trials to restart on the next track.

Each trail was composed of 8 tracks. Two in each of the four regions (DM1-only, DM1-merge, DM2-merge, DM2-only). Subjects were simply instructed to do their best to process as many of the incoming tracks as possible.

3.3.3 Results

Reliability: An initial examination of the data revealed that for two of the teams, performance on the slowest speed trials was less than 50%. Since a performance level of 33% can be obtained by answering randomly, the data from these teams was not included in the analysis. The performance of the other nine teams, for the slowest speed, was consistently high.

Overall Accuracy: Tables 3-1 through 3-4 summarize the proportion of tracks that were correctly processed for each of the trial speeds. The columns on the four tables are defined as follows.

- *DMi-only:* This column represents proportion correct in the DMi-only region.
- *Advice sent:* This column represents the proportion of tracks in the DMj-merge region for which the correct advice was sent to the other decision maker.
- *No merge:* For the tracks in the DMi-merge region for which advice was not received, this column represents the proportion of tracks for which the correct judgment was made based only on just the information in the Threat Information window.
- *Merged:* For the tracks in the DMi-merge region for which advice was received, this column represents the proportion of times the correct final judgment was made.

Table 3-1 Performance by Region, Speed = 400

Subject	DMi-only	Advice sent	No merge	Merged
TM1DM1	0.97	0.98	-	0.98
TM1DM2	0.97	0.98	-	0.68
TM2DM1	1	1	0	0.97
TM2DM2	0.97	0.98	-	0.97
TM4DM1	1	0.98	1	0.98
TM4DM2	0.95	0.97	0.5	0.94
TM5DM1	0.95	1	0.38	0.88
TM5DM2	0.7	0.77	0.22	0.75
TM6DM1	0.97	1	-	0.95
TM6DM2	1	1	1	0.98
TM7DM1	0.97	1	0.75	0.93
TM7DM2	0.95	0.92	0.1	0.86
TM8DM1	0.84	0.57	0.61	0.71
TM8DM2	0.78	0.57	0.3	0.57
TM9DM1	1	0.98	0.29	0.94
TM9DM2	0.98	1	1	0.98
TM10DM1	1	1	1	1
TM10DM2	0.98	1	0.8	0.93
Overall Result	0.943	0.929	0.446	0.916

Table 3-2 Performance by Region, Speed = 600

Subject	DMi-only	Advice sent	No merge	Merged
TM1DM1	0.86	0.95	0	0.96
TM1DM2	0.83	1	0.67	0.79
TM2DM1	1	0.82	0.65	0.84
TM2DM2	1	1	0.5	0.93
TM4DM1	1	0.97	0.6	0.98
TM4DM2	0.83	0.93	0.17	0.83
TM5DM1	0.88	0.97	0.34	0.92
TM5DM2	0.67	0.78	0.18	0.81
TM6DM1	0.82	1	-	0.9
TM6DM2	0.85	1	-	1
TM7DM1	1	1	0.69	0.94
TM7DM2	0.73	0.87	0.14	0.75
TM8DM1	1	0.97	0.74	1
TM8DM2	0.62	0.33	0.2	0.75
TM9DM1	0.97	0.88	0.32	0.71
TM9DM2	1	1	0.5	0.89
TM10DM1	0.98	1	0.17	0.96
TM10DM2	1	0.98	0.42	0.98
Overall Result	0.89	0.914	0.379	0.909

Table 3-3 Performance by Region, Speed = 750

Subject	DMi-only	Advice sent	No merge	Merged
TM1DM1	0.82	1	0	0.96
TM1DM2	0.73	1	1	0.81
TM2DM1	0.95	0.98	0.28	0.79
TM2DM2	0.98	0.98	0.33	0.98
TM4DM1	0.98	0.98	0.46	0.87
TM4DM2	0.86	0.93	0.38	0.83
TM5DM1	0.52	0.67	0.26	0
TM5DM2	0.53	0.78	0.29	0.8
TM6DM1	0.54	0.98	1	0.88
TM6DM2	0.32	1	0.67	0.98
TM7DM1	0.98	1	0.54	0.81
TM7DM2	0.72	0.9	0.29	0.73
TM8DM1	1	0.97	0.49	0.67
TM8DM2	0.78	0.23	0.32	0
TM9DM1	1	0.81	0.23	0.85
TM9DM2	1	0.9	0.26	0.82
TM10DM1	0.95	0.98	0.22	0.93
TM10DM2	0.88	1	0.25	0.91
Overall Result	0.808	0.907	0.327	0.881

Table 3-4 Performance by Region, Speed = 900

Subject	DMi-only	Advice sent	No merge	Merged
TM1DM1	0.48	1	0.26	0.94
TM1DM2	0.5	1	0.56	0.76
TM2DM1	0.77	0.98	0.33	0.67
TM2DM2	0.98	1	0.41	0.85
TM4DM1	0.9	1	0.28	0.6
TM4DM2	0.74	0.98	0.29	0.56
TM5DM1	0.37	0.73	0.29	
TM5DM2	0.47	0.78	0.26	0.33
TM6DM1	0.18	0.98	0.83	0.7
TM6DM2	0.08	1	0.43	0.98
TM7DM1	1	1	0.41	0.81
TM7DM2	0.6	0.9	0.32	0.8
TM8DM1	1	0.89	0.18	0
TM8DM2	0.75	0.42	0.25	1
TM9DM1	0.98	0.75	0.15	0
TM9DM2	0.98	0.72	0.32	0.33
TM10DM1	1	0.92	0.27	0.67
TM10DM2	0.93	1	0.3	0.7
Overall Result	0.69	0.889	0.295	0.766

Table 3-5 summarizes the average results for the four speed conditions. The performance in the No-merge group is largely irrelevant, since it reflects a condition were subjects did not receive an input they were expecting. Performance under this condition is expected to be low.

Table 3-5 Accuracy at Different Speeds

Speed	DMi-only	Advice sent	No merge	Merged
400	0.943	0.929	0.446	0.916
600	0.89	0.914	0.379	0.909
750	0.808	0.907	0.327	0.881
900	0.69	0.889	0.295	0.766

An examination of the other columns reveals that subjects consistently maintained high performance on the tracks where they needed to send advice. They also managed to maintain relatively high performance for the Merged judgments. Performance for the DMi-only judgments dropped more substantially as time available decreased. An detailed examination of the subject output files reveals that this is due to the order in which subjects processed the tracks. Under the lower time stress conditions, subjects would process tracks in the order recommended (i.e., send advice first, DMi-only tracks second, and then do the merge judgments). However, as time stress increased, subjects shifted the order. They would process the merge tracks before attempting to

process the DMi-only tracks. This is reflected in Table 3-6, which indicates the average rank order of processing of the tracks in different regions.

Table 3-6 Average Rank Order of Processing Tracks

	DMi-only		DMj-merge		DMi-merge	
Speed	Mean	S.D.	Mean	S.D.	Mean	S.D.
400	3.32	1.36	2.07	0.84	4.55	1.45
600	3.19	1.25	2.03	0.80	4.46	1.29
750	3.14	1.18	1.85	0.88	4.47	1.02
900	2.74	1.17	1.89	0.74	2.25	1.16

Vulnerable-to-bias judgments

The principal hypothesis was that vulnerable-to-bias decision procedures are executed less reliably than other decision procedures and are more vulnerable to the effects of time stress. This implies that performance for the FF, BB, SS and the BF, SF conditions will be higher than for the FB, FS and the BS, SB conditions. Tables 3-7 through 3-10 summarize performance under each condition for each level of time stress.

Table 3-7 Performance of Different Types of Merge Judgments, Speed = 400

	Not vulnerable to bias		Vulnerable to bias	
Subject	FF, BB, SS	BF, SF	FB, FS	BS, SB
TM1DM1	1	1	1	0.944
TM1DM2	1	0.545	0.417	0.611
TM2DM1	1	1	0.917	0.944
TM2DM2	1	1	1	0.889
TM4DM1	1	1	1	0.933
TM4DM2	1	1	1	0.833
TM5DM1	0.941	1	0.846	0.769
TM5DM2	0.733	0.7	1	0.625
TM6DM1	1	1	0.917	0.889
TM6DM2	1	1	1	0.944
TM7DM1	1	0.917	0.769	1
TM7DM2	0.941	1	0.636	0.857
TM8DM1	1	_	1	0
TM8DM2	1	0.667	0.5	0.25
TM9DM1	1	1	0.833	0.938
TM9DM2	0.941	1	1	1
TM10DM1	1	1	1	1
TM10DM2	1	1	0.917	0.813
Overall Result	0.9751773	0.93854749	0.88359788	0.86131387

In general, performance at each level supported the hypothesis that the vulnerable-to-bias decision procedures were executed less reliably than the other decision procedures. A statistical analysis of these results is reported in Table 3-11. This analysis was performed by examining the proportion of individual subjects for whom the relative performance results were in the predicted direction. For instance, for the prediction BF, SF > FB, FS at Speed = 400, there were nine subjects for which the average performance for the BF and SF condition was higher than for the FB and FS conditions, seven with equal performance, and one with performance in the reverse. A one-tailed sign test on a ratio of 9:1 gives the result $p < .05$.

Table 3-8 Performance of Different Types of Merge Judgments, Speed = 600

	Not Vulnerable-to-bias		Vulnerable-to-bias	
Subject	FF, BB, SS	BF, SF	FB, FS	BS, SB
TM1DM1	1	1	1	0.867
TM1DM2	1	0.786	0.643	0.765
TM2DM1	1	1	0.4	1
TM2DM2	1	1	0.786	0.938
TM4DM1	1	1	0.889	1
TM4DM2	0.875	0.714	1	0.75
TM5DM1	1	1	1	0.667
TM5DM2	1	1	0.6	0.75
TM6DM1	1	0.75	0.917	0.867
TM6DM2	1	1	1	1
TM7DM1	1	1	0.8	0.9
TM7DM2	0.571	0.75	0.857	0.833
TM8DM1	1	1	1	-
TM8DM2	0	1	1	1
TM9DM1	1	1	0	0.667
TM9DM2	1	0.875	0.778	0.923
TM10DM1	1	1	0.9	0.929
TM10DM2	1	1	1	0.923
Overall Result	0.97252747	0.93006993	0.84	0.88505747

It should be noted, however, that the results do not directly support the notion that the vulnerable-to-bias procedures degrade more rapidly than other procedures as stress increased. Although the FF, BB, SS and BF, SF conditions were more reliably executed than the FB, FS and the BS, SB

Table 3-9 Performance of Different Types of Merge Judgments, Speed = 750

	Not Vulnerable-to-bias		Vulnerable-to-bias	
Subject	FF, BB, SS	BF, SF	FB, FS	BS, SB
TM1DM1	1	1	0.875	0.938
TM1DM2	1	0.667	0.583	0.882
TM2DM1	1	1	0.5	0.667
TM2DM2	1	1	0.875	1
TM4DM1	1	1	0.75	0.8
TM4DM2	1	1	-	0.667
TM5DM1	-	-	-	0
TM5DM2	0.667	-	-	1
TM6DM1	0.941	1	0.917	0.722
TM6DM2	1	1	1	0.941
TM7DM1	1	1	0.571	0.733
TM7DM2	1	0.667	0	1
TM8DM1	0.5	1	-	-
TM8DM2	-	-	0	-
TM9DM1	1	1	0.5	0.8
TM9DM2	1	1	0.667	0.667
TM10DM1	1	1	0.667	0.889
TM10DM2	1	1	0.857	0.778
Overall Result	0.97916667	0.94565217	0.74444444	0.82894737

conditions, the difference in performance between these conditions did not increase consistently as time stress increased. However, this lack of support is an artifact of the experimental design, since it **must** be the case that in non time stressed situations subjects could flawlessly execute the simple decision procedures in this experiment.

Comparison to Pilot Study

A pilot study of four teams was run prior to executing the main experiment. The pilot procedure differed from the main experiment in that there was more training, subjects went through more sessions, and the slowest speed was considerably slower. In the pilot study, all four groups attained near perfect performance at the slowest speeds. The other results were about the same.

Table 3-10 Performance of Different Types of Merge Judgments, Speed = 900

Subject	Not Vulnerable-to-bias		Vulnerable-to-bias	
	FF, BB, SS	BF, SF	FB, FS	BS, SB
TM1DM1	1	0.889	1	0.875
TM1DM2	1	0.667	0.571	0.733
TM2DM1	1	0.5	-	-
TM2DM2	1	1	1	0.667
TM4DM1	0.75	0.667	0.667	0
TM4DM2	1	1	0.25	0.5
TM5DM1	-	-	-	-
TM5DM2	1	0.25	-	0
TM6DM1	0.933	0.833	0.545	0.5
TM6DM2	1	1	0.9	1
TM7DM1	0.8	1	0	0.778
TM7DM2	-	0	1	1
TM8DM1	-	0	-	0
TM8DM2	-	1	-	1
TM9DM1	-	0	0	0
TM9DM2	-	1	0	0
TM10DM1	0.5	1	0	0.667
TM10DM2	1	0.75	1	0.333
Overall Result	0.94285714	0.7625	0.67307692	0.68539326

Table 3-11 Number of Subjects for Which Vulnerable-to-Bias Procedures had more Errors than the Non Vulnerable-to-Bias Procedures

	Predicted Direction of Effect											
	FF,BB,SS=BF,SF			BF,SF>FB,FS			BF,SF>BS,SB			FB,FS=BS,SB		
	Observed			Observed			Observed			Observed		
Speed	>	=	<	>	=	<	>	=	<	>	=	<
400	4	10	3	9	7	1*	13	2	2**	11	2	5
600	5	11	2	10	5	3*	9	4	4	5	2	9
750	2	11	2	12	1	0***	11	1	2*	3	1	9
900	7	3	2	7	3	3	10	4	2*	5	3	5

* $p < .05$ (one-tailed sign test)
** $p < .01$
***$p < .001$

3.3.4 Discussion

Overall, the results support the hypothesis that vulnerable-to-bias decision procedures are less reliably executed than other decision procedures. Time available ranged from 45 seconds to 20 seconds. For all times available, the vulnerable-to-bias decision procedures were the less reliably executed procedures. This result occurred even though the decision procedures in which subjects were trained were *very* simple; involving only a few rules and no more than two preconditions per rule.

A possible confounding variable is the possibility that subjects adapted to time stress by routinely ignoring the other DMs' advice. Although this behavior be would consistent with the vulnerable-to-bias hypothesis, it would have the effect of reducing the effective sample size. Rather than treating each track as an independent sample, each set of tracks becomes a sample of one strategy shift. However, the results do not support this explanation. If true, this explanation would predict that performance for the BF, BS conditions would be the same as the BB, FF, SS condition, and that performance for the FB, FS and BS, SB condition would be near zero. This later result clearly did not occur. Instead, the results suggest that subjects adapted to severe time stress by changing the order in which they processed tracks. For the lower stress conditions, subjects ordered tracks in the way they were trained; by first sending advice (DMj-merge), then taking care of the their own tracks (DMi-only), and finally by processing the tracks where they had to merge their judgment with the other DM's advice (DMi-merge). This was the optimal strategy since it (1) guaranteed that the other DM would have time to send advice, and (2) it left the most time consuming tracks to be processed last. Interestingly, as time stress increased, subjects adapted by moving away from the optimal strategy; processing tracks in the order DMj-merge, DMi-merge, and DMi-only. Indeed, this order was the worst possible order for processing tracks under time stress, since the first tracks to be completely processed (DMi-merge) are also the tracks that take the longest to process. One possible explanation for this maladaptive behavior is that subjects responded to time stress by processing the tracks by "visual order", i.e., moving from upper left to the lower right portion of the screen. They were reducing cognitive workload by simplifying the decision procedure for selecting which track to process next.

Overall these results suggest that when specifying team decision procedures, vulnerable-to-bias procedures should be avoided. Although a well trained team may execute vulnerable-to-bias procedures reliably in non stress conditions, these procedures are very vulnerable to the effects of time stress. In addition, these results point to the importance of procedures for adapting to stress. The subjects in this experiment adapted inappropriately.

CHAPTER IV

DERIVING ARCHITECTURES

4.1 INTRODUCTION

In the previous two chapters, the derivation of organizational decision strategies and their allocation to organizational structures were described. The procedures were illustrated with very small organizations - organizations consisting of several decison makers. While this is a realistic model of a team, it must be recognized that teams operate in the context of larger organizations. Teams are part of larger organizational structures which, both in the military, in industry, and in academial have a hierarchical structure. While the functional architecture of such an organization may not be hierarchical, its organizational architecture usually is. A group of organization members constitute a team; then a group of teams - with or without additional organization members who provide leadership and coordination - forms a unit at the next level; a group of these units form a unit at the next higher level, etc. The classic military paradigm of soldiers organized into squads, squads into platoons, platoons into batallions, batallions into regiments or brigades, then divisions, corps, and armies is pervasive in organizational design, even though the actual organization may not function in exactly that way. This paradigm is useful, however, in extending the results obtained over the last few years that apply to small teams to the case of large organizations. The concept of a building block can be used to contruct organizations of arbitrary size and complexity. For example, at the lowest level, several types of teams with specific properties and features can be designed. The interactions among the organizations members can be specified using the model of the interacting decision maker. Now each one of these teams can be aggregated into a single model that is analogous to the single decision maker model. One can then design the interactions among teams using a variation of the exsiting algorithms. One can aggregate again and move to a higher level. This conceptual process leads to the utilization of the concept of "level of abstractions." One can see an organization at different levels of abstraction. Each level leads to a different representation of the organization, but all theset representations share common features. One feature in particular that is critical for performance is the coordination among the components of the organization as seen at each level of abstraction, especially when the organizations have variable structure. These are the subjects of this chapter.

4.2 ALGORITHMIC DESIGN OF DISTRIBUTED DECISION MAKING ORGANIZATIONS

4.2.1 Introduction

Distributed Decision Making Organizations (DMOs) are defined as those systems in which the capacity for reasoning is dispersed across its component subsystems.(Levis, 1984) In a distributed system, each function is spread over a number of nodes so that each node's activity contributes a little to each of several different functions. The systems characterized as DMOs carry out a number of functions, sometimes in sequence and sometimes concurrently. Thus, the problem of decomposing functions and allocating the decomposed functions to available resources is not a simple one. The allocation of several decomposed functions to different nodes must be done in such a manner that the resulting organizational structure do not violate a number of structural and cognitive constraints.

A quantitative methodology for modeling, designing and evaluating distributed decision making organizations has been developed at the MIT Laboratory for Information and Decision Systems by Remy (1986), Andreadakis (1988), and Demaël (1989). In this work, an organization is considered as a system performing a task; the system is modeled as an interconnection of organization nodes (Decision Making Units, DMUs). Each organization member is represented by a multi-stage model. The origin of this multi-stage model can be traced back to the four stage model of the interacting decision maker with bounded rationality introduced by Boettcher and Levis (1982). The formal specification of the allowable interactions between decision makers was made by Remy (1986), which led to an algorithm. The Lattice algorithm generates all feasible fixed structure architectures that meet a number of structural and user-defined constraints. Andreadakis and Levis (1988) introduced an alternative model that was not based on the decision maker model, but on the function carried out by a resource, whether that represented a human or a machine. While this was a five stage model, it was very similar to the four stage one in terms of the allowable interactions. That model formed the basis for a different algorithm for organization design, the Data Flow Structure (DFS) algorithm (Andreadakis and Levis, 1988). In this approach to organization design, the required data flow structures for the system are determined first and then functions are assigned to resources. In a parallel effort, (Monguillet and Levis, 1993) formalized the notion of variable structure decision making organizations. Demaël and Levis (1990) extended the earlier work and developed a methodology for modeling and generating variable structure distributed decision making organizations. They presented a mathematical framework for modeling systems that adapt their structure of interactions to the input they process.

The various models mentioned have been used to address a number of problems in the design, analysis, and evaluation of distributed decision making organizations supported by decision aids and decision support systems. Levis (1992) reassessed the various models and their variants and concluded that a slightly more general model can subsume all previous ones without invalidating any of the cognitive modeling or the design algorithms. The design methodology presented in this paper uses this generalized five stage model.

All previous efforts mentioned above resulted in methodologies for designing *flat* DMO architectures in which the system is viewed only from a single level of detail. When it comes to *multilevel* DMOs, these methodologies are confronted by the combinatorial nature of the problem. On the other hand, complex distributed decision making organizations are characterized by the hierarchical arrangement of their subsystems, i.e., the organization of an army is done in terms of soldiers and officers, squads, platoons, companies, battalions, and so on. The DMOs are described by families of structures with each family concerned with the behavior of the system as viewed from a different level of abstraction, i.e., the army's organizational structure can be viewed in terms of interactions among battalions, or companies, or individual soldiers and officers - the most detailed description of the organization. This report presents a mathematical model of interactions among sub-organizations defined at different levels of abstraction. The model requires the designer to determine first the level of at which he will consider the organization. The organizational units at different levels are specified in terms of their constituent organizational units. At the lowest level, the decision making unit is a human decision maker represented by a five stage model (Levis, 1992). At all other levels, the decision making units are organizations in their own right. Depending on the particular level chosen, the designer is required to characterize with an arbitrary degree of precision the class of interactions among the decision making units comprising the organization as seen at that level. The specificity of the designer's requirements determines the degrees of freedom left. Lattice theoretic results are used to define a partial order among all allowable organizational structures. The solution set, then, can be characterized by its boundaries; this is an extension of the results in Remy and Levis (1988).

The mathematical formulation of the problem is based on Petri Net theory. All the allowable structures are translated into Petri Net representations. The set of all allowable organizational structures can then be analyzed and a particular organizational structure can be chosen as a result of a comparison of performance with respect to some designer-defined criteria. The entire organization is described in terms of its decision making units. The organizational structures associated with these units are *folded* or *unfolded* to represent the system's architecture at different levels. A set of connectivity rules are formulated to translate interactions among units of the

organization defined at a given level to their lower level representations. The interactions that exist at a higher level of abstraction are translated to their more detailed description whenever an organization is unfolded to a more detailed representation. The connectivity rules are based on the concept of a multi-echelon hierarchy; the hierarchical relationships are formulated on the basis of messages that flow to and from the decision making units.

The design methodology (Zaidi, 1991) is illustrated in this report with a hypothetical organization design. Since the results are based on Hierarchical Petri Net theory, a brief review is presented in the following section.

4.2.2 Hierarchical Petri Nets

The basic concepts of ordinary Petri Nets are not presented in this section. Introductory material on Petri Nets can be found in Peterson (1981), Brams (1983), Reisig (1985), or Murata (1989).

Hierarchical Petri Nets allow the designer to create a large model composed of many sub-models, and isolate a segment to study its details without disturbing or altering the entire structure. They also provide a modular approach towards modeling a complex system. This feature is vital for designing complex systems that require frequent study of alternative structures during the development process. The hierarchical nature of the Petri Nets provides the designer an abstraction mechanism that

- provides an overview and an adequate representation of system structure, absent in single level system models;
- hides details in a consistent way;
- separates into well-defined and reusable components; and
- supports top-down and bottom-up design strategies.

Compound Transition

If a sub-net of a Petri Net model is replaced by a single transition, the single transition is termed *compound transition.* It represents the aggregated effect of the processes represented by the transitions of the sub-net. The system with compound transitions describes the system at a higher level of abstraction than the one without them.

Figure 4.1 shows a hypothetical Petri Net model of a system in which the system's functionality is described at the most detailed level. The transitions shown in the figure represent the processes and

algorithms carried out by them. The dotted box contains the processes that are to be aggregated. In Figure 4.2, the outlined sub-net is shown replaced by a single transition - a compound transition denoted by the label "HS". The sub-net that represents the compound transition at a *sub-page* is shown in Figure 4.3. The term sub-page is used in *Design/CPN™*, a commercially available software package for Hierarchical Petri Nets (*Design/CPN:A Reference Manual*, 1991) to denote pages which contain the sub-nets replaced by compound transitions and compound places.

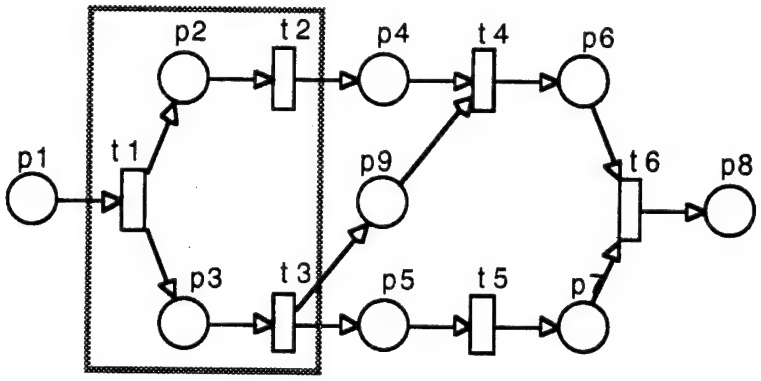


Figure 4.1 Detailed Description of a System

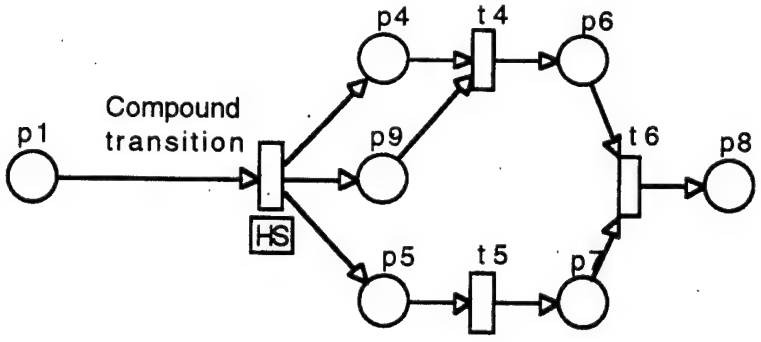


Figure 4.2 System's Description with a Compound Transition

In Figure 4.3, the places with label "B in" or "B out" represent the *port nodes*. Port nodes are defined to be the input and output places of the sub-net; they are its connections with the uncompound net. On the other hand, all those places whose input and output transitions are defined within the sub-net are not port nodes. Port nodes are used to preserve the connectivity of the original net. They model the sockets for the places that exist in the preset and postset of the compound transition in the system's net. The places p1, p4, p5, and p9 in Figure 4.2 are defined as port nodes in Figure 4.3.

When it is desired to replace a sub-net by its compound transition representation care must be taken in selecting the boundaries of the sub-net. In order to replace a sub-net of a net by a compound transition, the boundaries of the sub-net should be comprised only of transitions. The boundary of a sub-net is defined to be the set of nodes belonging to the sub-net having at least one of their input and/or output nodes be nodes of the net that do not belong to the sub-net. A sub-net with at least one place at the boundary of the sub-net can not be replaced by a compound transition. On the other hand, if a sub-net of a Petri Net model is replaced by a single place, the single place is termed *compound place*. It represents the aggregated effect of the sub-net replaced by the compound place.

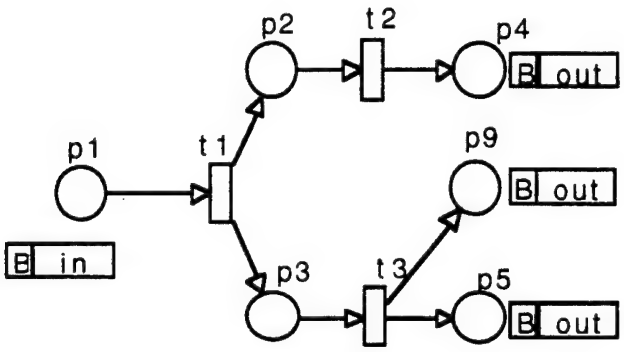


Figure 4.3 Sub-page Representation of the Compound Transition

Folding and Unfolding a Net

A Petri Net model of a system is said to be folded, if certain sub-nets of the net are aggregated by compound transitions and/or compound places. The folded net obtained as a result describes the system at a higher level of abstraction. The sub-nets replaced by compound transition and/or compound places are moved to the sub-pages as a result of folding the net. The original detailed description of the system net can be retrieved by uncompounding the compound transitions and compound places, i.e., by moving the sub-nets back to their original locations. A compound transition or a compound place, therefore, represents a sub-net stored at a sub-page with port nodes to preserve the original connectivity of the net. The process of uncompounding all the compound transitions and compound places is termed unfolding the net. In this work, the organizational structures represented in terms of Petri Nets are folded by creating compound transitions representing different sub-organizations. *The processes of folding and unfolding do not affect the Petri Net properties of the structures; the structures obtained as a result of folding and unfolding are legitimate, executable, Petri Nets.* The folded structures can be executed with or without the sub-page structures. Figure 4.4 presents a Petri Net with two of its sub-nets outlined by dotted boxes. The outlined sub-nets are replaced by their compound transition representation in

Figure 4.5. The Petri Net in Figure 4.5 is the folded version of the net in Figure 4.4. It represents the same system in Figure 4.4 but at a higher level of abstraction.

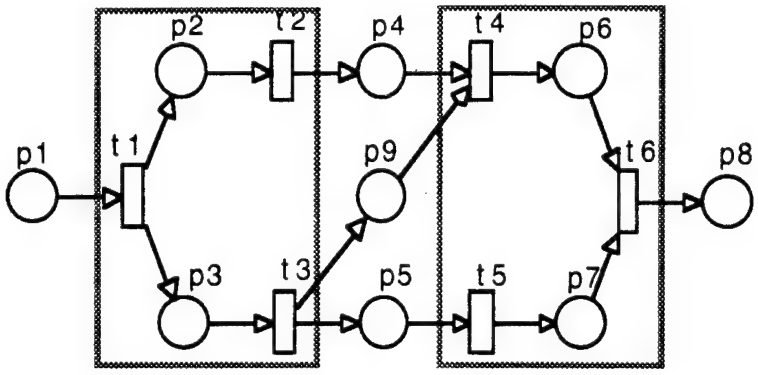


Figure 4.4 Petri Net of a System

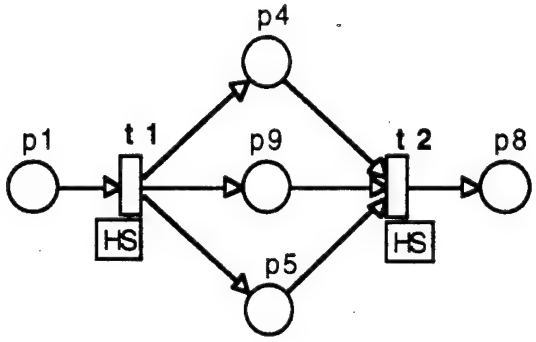


Figure 4.5 Folded Petri Net

The sub-nets that are moved to sub-pages as a result of folding are shown in Figs. 4.6 and 4.7. Figure 4.6 represents the net replaced by compound transition **t1** along with the port nodes, while the sub-net replaced by the compound transition **t2** is shown in Figure 4.7.

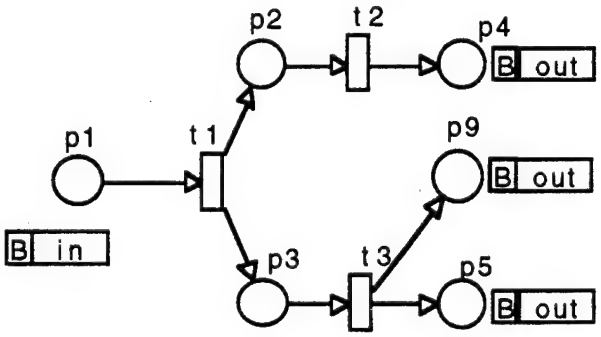


Figure 4.6 Sub-net Replaced by Compound Transition **t1**

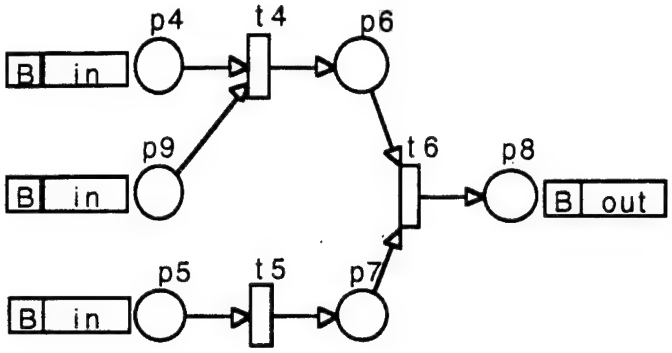


Figure 4.7 Sub-net Replaced by Compound Transition **t2**

The places p4, p5, and p9 in Figure 4.5 are all the output places of the compound transition **t1** and input places of compound transition **t2**. If the system's behavior at a higher level of abstraction is desired to be depicted, the three places p4, p5, and p9 can also be represented by an equivalent single place **p2** with input and output arcs having a weight of 3 as shown in Figure. 4.8. If the single equivalent place **p2** models the flow of information from the aggregated processes represented by **t1** to aggregated processes represented by **t2** and the three places between **t1** and **t2** in Figure 4.5 represent a redundancy in the flow of information since the tokens are defined to be indistinguishable, then Figure 4.9 may be used where there is no weighting on the input and output arcs of **p2**.

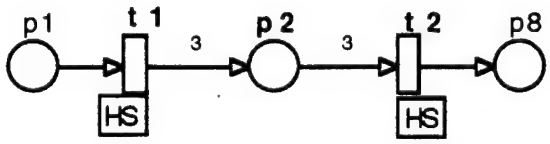


Figure 4.8 Folded Version of the Net in Figure 4.4

The net in Figure 4.9 can be unfolded to the net in Figure 4.4 by uncompounding the compound transitions **t1** and **t2**. The places that are represented by the equivalent place are defined in the sub-nets in Figures 4.6 and 4.7; therefore, whenever the compound transitions are uncompounded, all the places present in the original net will be retrieved from the sub-pages producing the original detailed description of the net in Figure 4.4.

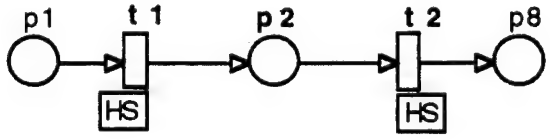


Figure 4.9 Folded Version of the Net in Figure 4.4

The folding process described in this section will be used in the design methodology presented in this report. The process of folding Petri Nets also refers to a technique used to translate Ordinary Petri Nets to their Colored Petri Net representations. Since Colored Petri Nets are not used here, that folding process is not discussed; interested readers are referred to Jensen (1992).

4.2.3 Mathematical Model

The formal concepts of multilevel, hierarchical systems are defined in Mesarovic et. al. (1970). They introduced the concept of *stratum* for modeling organizational architectures when viewed from different levels of detail. The formal definition of a Stratified Decision Making Organization follows:

Definition 4.1: A Stratified Decision Making Organization (SDMO) is defined to be a Decision Making Organization (DMO) in which a unit (or system) on a given stratum is a component unit (or subsystem) on the next higher stratum. In a SDMO, Decision Making Units (DMU) can be either Decision Making Sub-Organizations (DMSOs) or human Decision Makers (DMs) depending upon the level of abstraction used to represent the organizational structure of the DMO.

The example SDMO is presented in Figure 4.10. The nodes shown by boxes are DMUs comprising the SDMO at different levels of abstraction. In the three-strata SDMO, the highest stratum, stratum 0, contains only one organizational structure, which represents the highest level of abstraction that can be used to describe an organization. In stratum 1, the SDMO is described in terms of the interactions among three DMUs shown in the figure. Each DMU in stratum 1 itself is comprised of two DMUs, as shown in stratum 2. Therefore, the n^{th} (2^{nd}) stratum description of the SDMO represents an elaborated and detailed structure of the interactions among DMUs in stratum n. The range of n is defined as $1 \leq n \leq N$, where N represents the lowest possible stratum at which the DMUs can not be decomposed further. The determination of the value of N is application dependent, i.e., in human organizations, the DMUs at stratum N are human Decision Makers (DM). All nodes are labeled by an alphanumeric code, DMU_{ik}, where i represents the node number at stratum k. The set of all nodes at stratum k contains $|\mu_k|$ elements, i.e., $\mu_k = \{1, 2, ..., |\mu_k| \}$ and $i \in \mu_k$.

A DMU at stratum k, where $1 \leq k < n$, is defined as a *compound node* (Zaidi, 1991). A compound node is a decision making sub-organization (DMSO) comprised of a number of DMUs defined at the next lower stratum.

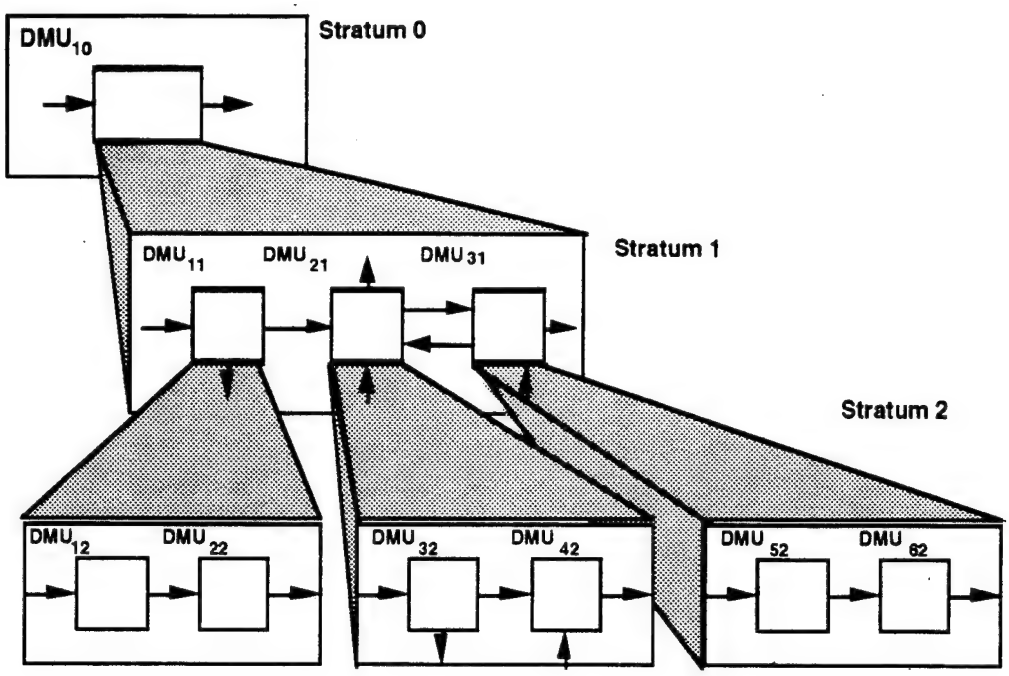


Figure 4.10 Three-Strata Organization

Definition 4.2: A compound node is a *folded* structure of the lower-strata DMUs and their interconnections.

The five stage model of a compound node is presented in Figure 4.11. The labels SA, IF, TP, CI and RS are generic names for the *situation assessment, information fusion, task processing, command interpretation,* and *response selection* processes, respectively. (Levis, 1992) The suffix C represents the compound node notation of these processes. The figure also shows all the input and output stages of the compound node. The five stage model and the input/output interactional structure of a compound node are identical to the five stage model of a human DM. The physical interpretation of these interactions, however, varies slightly from that of a single DM. All organizational structures defined in any arbitrary stratum can be folded to their compound node representations, a fact that led to the definition of a compound node. (Zaidi, 1991)

A compound node receives input or data x from the external environment (sensors) or from other compound nodes of a system. The incoming data are processed in the compound situation assessment (SAC) stage to get the assessed situation z. This variable may be sent to other compound nodes. If the compound node receives assessed data from other compound nodes, these data z' are fused together with its own assessment z in the compound information fusion (IFC) stage to get the revised assessed situation z". The assessed situation is processed further in the

compound task processing (TPC) stage to determine the strategy to be used to select a response. The variable v contains both the assessed situation and the strategy to be used in the compound response selection stage. A particular compound node may receive a command v' from super-ordinate compound nodes. This is depicted by the use of the compound command interpretation (CIC) stage. The output of that stage is the variable *w* which contains both the revised situation assessment data and the response selection strategy. Finally, the output or the response of the compound node, *y*, is generated by the compound response selection (RSC) stage.

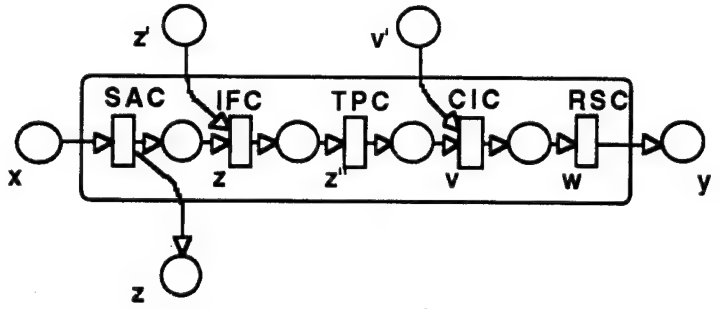


Figure 4.11 Compound Node

Only certain types of interactions make sense within the model. (Remy and Levis, 1988; Zaidi, 1991) They are depicted in Figure 4.12. For the sake of clarity, only the links from the i^{th} DMU to the j^{th} DMU are presented. The symmetrical links from j to i are valid interactions as well.

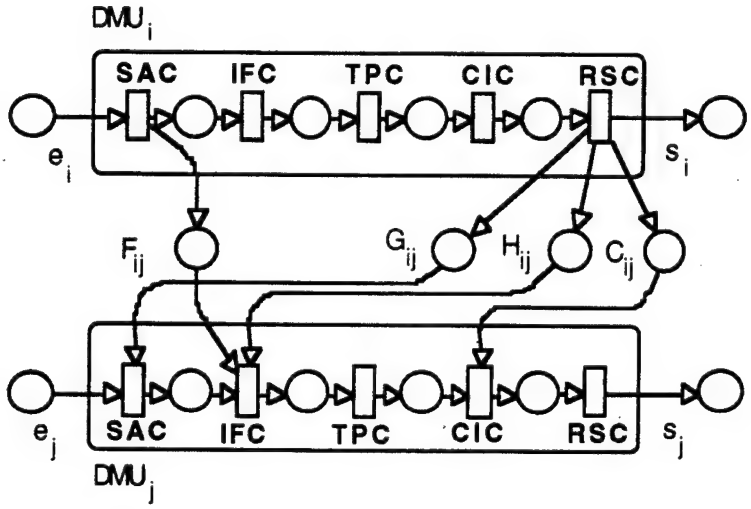


Figure 4.12 Allowable Interactions Between two DMUs

The binary variable e_i represents the *external input* to a decision making compound node. The presence of such a link characterizes the fact that a particular DMU may receive data from the external environment or from another DMU located at the next higher stratum. The binary variable s_i represents the *external output* of a decision making compound node to processes external to the organizational structure considered. The binary variable F_{ij} depicts the transmission of assessed situation from compound node i to compound node j; G_{ij} models the transmission of control from the output of a decision making compound node to the input of another; H_{ij} models the result or processed information sharing type of interaction between two decision making compound nodes; and C_{ij} represents the flow of instructions or commands from one decision making compound node.

Proposition 4.1: Every Decision Making Unit (DMU) can be represented by the five stage model shown in Figure 4.11, regardless of the stratum in which it is defined.

The proof of the proposition follows from the folding procedure. If all the situation assessment stages of an organizational structure (in stratum n) are compounded together into a single compound situation assessment stage and a similar folding procedure is employed for all similar stages, one would obtain the five stage model of Figure 4.11. The five stage model so obtained will represent the same organizational structure at a higher level of abstraction (stratum n-1). For a detailed description of this folding procedure, refer to Zaidi (1991).

The variables e_i, s_i, F_{ij}, G_{ij}, H_{ij}, C_{ij} in Figure 4.12 are binary variables taking values in {0, 1}, where 1 indicates the presence of the corresponding link in the organizational structure at the stratum for which the structure is defined. Note that the value of the variable does not indicate the number of such links which actually exist. The variables are aggregated into two vectors $\boldsymbol{e}$ and $\boldsymbol{s}$, and four matrices $\boldsymbol{F}$, $\boldsymbol{G}$, $\boldsymbol{H}$, and $\boldsymbol{C}$.

The DMUs of the compound node are defined in stratum k+1. The structure of the compound node i at stratum k will be the five stage model shown in Figure 4.11. Σ_{ik+1} represents the interactional structure of the compound node i, when the level of abstraction used to describe the structure is of stratum k+1. The compound node i itself is defined as a DMU for stratum k.

The interaction structure of a compound node i, $i \in \mu_k$ consisting of m DMUs is represented by the following tuple.

$$\Sigma_{ik+1} = \{ \boldsymbol{e}, \boldsymbol{s}, \boldsymbol{F}, \boldsymbol{G}, \boldsymbol{H}, \boldsymbol{C} \} \qquad i \in \mu_k \qquad k = 0,1,2,...,n$$

where ***e*** and ***s*** are m x 1 arrays representing the interactions of the m-DMUs.

$$\boldsymbol{e} = [e_a] \quad \boldsymbol{s} = [s_a] \qquad a = 1,2,...,m \qquad m \in \mu_{k+1}$$

F, G, H, and ***C*** are four m x m arrays representing the interactions among the DMUs of the organizational structure represented by compound node i.

$$\boldsymbol{F} = [F_{ab}] \quad \boldsymbol{G} = [G_{ab}] \quad \boldsymbol{H} = [H_{ab}] \quad \boldsymbol{C} = [C_{ab}] \qquad b = 1, 2, ..., m \qquad m \in \mu_{k+1}$$

The diagonal elements of the matrices ***F, G, H,*** and ***C*** are set identically equal to zero; DMUs are not allowed to interact with themselves.

$$F_{aa} = G_{aa} = H_{aa} = C_{aa} = 0 \qquad a = 1, 2, ..., m \qquad m \in \mu_{k+1}$$

4.2.4 Design Requirements

The interactional requirements for each of the compound nodes in a multilevel organization in terms of its lower stratum DMUs can be translated into requirements on the arrays. The designer may rule in or rule out some of the links by putting 1's and 0's at corresponding places in the arrays. This introduces the notion of *user-defined* constraints (R_u).

In the illustrative example, Figure 4.10, the interactional structures of the compound nodes in strata 1 and 0 are defined in terms of the nodes/compound nodes in strata 2 and 1 respectively.The user-defined constraints for DMU_{11}, in terms of DMU_{12} and DMU_{22}, are given as the tuple Σ_{12}.

$$e = [1 \quad x] \qquad F = \begin{bmatrix} 0 & 1 \\ x & 0 \end{bmatrix} \qquad G = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$s = [0 \quad x] \qquad H = \begin{bmatrix} 0 & x \\ x & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

The user-defined constraints for DMU_{21}, in terms of DMU_{32} and DMU_{42}, are given as the tuple Σ_{22}.

$$e = [1 \quad 1] \qquad F = \begin{bmatrix} 0 & x \\ x & 0 \end{bmatrix} \qquad G = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$s = [1 \quad 1] \qquad H = \begin{bmatrix} 0 & x \\ x & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

The user-defined constraints for DMU_{31}, in terms of DMU_{52} and DMU_{62}, are given as the tuple Σ_{32}.

$$e = [1 \quad x] \qquad F = \begin{bmatrix} 0 & x \\ 0 & 0 \end{bmatrix} \qquad G = \begin{bmatrix} 0 & x \\ x & 0 \end{bmatrix}$$

$$s = [x \quad 1] \qquad H = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

The user-defined constraints for DMU_{10}, in terms of DMU_{11}, DMU_{21} and DMU_{31} are given as the tuple Σ_{11}.

$$e = [1 \quad 1 \quad 1] \qquad F = \begin{bmatrix} 0 & x & 0 \\ x & 0 & x \\ 1 & 0 & 0 \end{bmatrix} \qquad G = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$s = [1 \quad 1 \quad 1] \qquad H = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & x \\ 0 & 0 & 0 \end{bmatrix}$$

The x's in the arrays represent the unspecified elements or optional links. The optional links determine the degrees of freedom left in the design process, and potentially yield a number of candidate solutions to the design problem, all satisfying the user-defined constraints (R_u).

4.2.5 Structural Requirements

The degrees of freedom left in the design procedure result in a very large set of organizational structures for each compound node. However, a number of them may correspond to patterns of interactions among DMUs that do not make physical sense. This leads to the definition of structural constraints (R_s). The structural constraints are divided into two sets as follows:

- *Global Constraints:* The set of constraints that must be satisfied by all the organizational forms regardless of the stratum for which they are defined.

- *Compound Node Constraints :* The set of constraints that are defined only for those organizational forms which have compound nodes as DMUs.

Let Σ_{qk} be the organizational form in stratum k defined for node q in stratum k-1 with DMUs i and j being the compound nodes. Then the fixed structure associated with Σ_{qk} must satisfy the following constraints.

Global Constraints

(R1) The Ordinary Petri Net that corresponds to Σ_{qk} should be connected, i.e., there should be at least one (undirected) path between any two nodes in the net. A directed path should exist from the source place to every node of the net and from every node to the sink.

(R2) The Ordinary Petri Net that corresponds to Σ_{qk} should have no loops. i.e., the structure must be acyclic.

(R3) In the Ordinary Petri Net that corresponds to Σ_{qk}, there can be at most one link from the RS/RSC stage of a DMU i to another DMU j, i.e., for each i and j, only one element of the triplet $\{G_{ij}, H_{ij}, C_{ij}\}$ can be non-zero. The analytical expression of this constraint is given as:

$$\forall\ (i, j) \in [1..|\mu_k|]^2 \qquad G_{ij} + H_{ij} + C_{ij} \leq 1 \qquad i \neq j$$

(R4) Information fusion can take place only at the IF/IFC and CI/CIC stages. Consequently, the SA/SAC stage of a DMU can either receive information from the external environment, or an output from another DMU. The translation of this constraint into mathematical terms follows:

$$\forall\ j \in [1..|\mu_k|] \qquad e_j + \sum_{i=1}^{m} G_{ij} \leq 1$$

The first part of constraint R1 eliminates any organizational form that does not represent a single structure. The second part of R1 insures that the flow of information is continuous within the organizational structure. It eliminates internal sink or source places. Constraint R2 allows acyclical organizational structures only. This restriction is imposed to avoid deadlocks and infinite circulation of messages within the organization. Constraint R3 indicates that it does not make sense to send the same signal to the same DMU at several stages. It is assumed that once the signal has been received by a DMU, it is stored in its internal memory and can be accessed at later stages.

Constraint R4 has to do with the nature of the IF/IFC stage. The IF/IFC stage has been introduced explicitly to fuse the situation assessments from other DMUs. It prevents a DMU from receiving more than one input at the SA/SAC stage (Zaidi, 1991).

Compound Node Constraints

(C1) In the Petri Net that corresponds to Σ_{qk}, there must be an input link to the SAC stage of a DMU i. This input link can be an input from the external environment or an output from another DMU. The analytical expression of the constraint is given as:

$$\forall\ j \in [1..|\mu_k|] \qquad e_j + \sum_{i=1}^{m} G_{ij} = 1$$

(C2) In the Petri Net that corresponds to Σ_{qk}, there must be at least one output link from the RSC stage of a DMU i. This output link can be an output to the external environment or to another DMU j, or both. The analytical expression is given as:

$$\forall\ j \in [1..|\mu_k|] \qquad s_j + \sum_{i=1}^{m} G_{ji} \geq 1$$

Constraint C1 insures an input connection to a compound node DMU. Constraint C2 insures an output connection to a compound node DMU. The constraint enforces the presence of the RSC stage of a compound node. Once the SAC and RSC stages are present, all the intermediate stages must also be present, thus realizing the fact that all the stages should appear in a compound node structure. The application of constraint R1 on organizational forms with compound nodes as DMUs implies constraints C1 and C2.

4.2.6 Convexity Of Constraints

Definition 4.3: A property S defined on A is convex if and only if every element x of A located in the interval [a, b], where a and b satisfy S, also satisfies S.

Proposition 4.2 (Remy, 1986): If a property S is convex on A, a convex subset A1 of A that satisfies S is completely characterized by its minimal and maximal elements as:

$$A1 = \{x \in A1 \mid \exists\ (a1, b1) \in A1_{min} \times A1_{max}\ \ a1 \leq x \leq b1\ \}$$

If a set is convex, its structure can be assessed with three simple tools, a partial ordering, a set of minimal elements and a set of maximal elements. Any element that is below one maximal element and above one minimal element belongs to the set. There is no need for an extensive, and possibly combinatorial, description of all the elements. Finding convex subsets in the set of nets defined by

the tuple Σ is quite important since convexity allows the description of the subsets without resorting to a combinatorial computational problem. In that case, the set of solutions can be obtained in terms of the minimal and maximal elements of the set. The constraints R are properties on the set of nets defined by the tuples Σ, since a constraint is either satisfied or violated by a given structure.

Proposition 4.3: Constraints R_u are convex.

Proof: The specifications defined by R_u are realized by placing 1s and/or 0s at the appropriate places in the arrays ***e, s, F, G, H,*** and ***C*** in order to rule in and/or rule out certain interactional links between DMUs. Let Σ'_{ik+1} and Σ''_{ik+1} be two elements of the set of nets satisfying constraints R_u, then a net Σ_{ik+1} located in the interval $[\Sigma'_{ik+1}, \Sigma''_{ik+1}]$ will also satisfy these constraints since the addition or removal of all other links except the ones placed by the user do not have any effect on these constraints.

Proposition 4.4 : The constraints R2, R3, R4 defined on the set of nets Σ are convex.

Proof: Let us consider R2. If a net defined by a tuple Σ is acyclical, i.e., fulfills R2, then any net obtained by removing links from the initial net will also be acyclical. Loops can not be created in a loop-free structure by removing links. The same argument applies to the constraints R3 and R4. For a detailed proof of the proposition for constraints R3 and R4, see Remy, 1986, and Demaël, 1989.

Proposition 4.5: Constraint R1 defined on the set of nets Σ is not convex.

Proof: The constraint R1 is not convex since it is possible to break the connectivity of a fixed structure by removing a link as well as by adding a link. This happens, for example, if a link that is added to the structure originates from a transition of the current net but does not terminate at a transition that was previously in the net. In that case, a transition without output place is created, which violates **R1.**

Proposition 4.6: Constraints C1, C2 are convex

Proof: The restrictions imposed by C1 and C2 are realized by placing 1s at the appropriate places in the arrays ***e***, ***s***, and ***G*** in order to ensure that a compound node structure has both input and output links. Let Σ'_{ik+1} and Σ''_{ik+1} be two elements of the set of nets satisfying constraints C1 and C2, then a net Σ_{ik+1} located in the interval $[\Sigma'_{ik+1}, \Sigma''_{ik+1}]$ will also satisfy these constraints since

the addition or removal of all other links except the ones placed by C1 and C2 do not have any effect on these constraints.

4.2.7 Computation Of Solutions

It can be easily inferred from the discussion in the previous section that the set of nets that satisfy the user-defined constraints (R_u), denoted as $W(R_u)$, is a lattice (Birkhoff, 1948), and therefore, can be characterized by its boundaries, the Universal and the Kernel Nets (Remy and Levis, 1988).

Definition 4.3: Universal and Kernel Nets: The Universal Net associated with the constraints R_u - $\Omega(R_u)$ - is the net defined by the tuple $\sum$ obtained by replacing all undetermined elements of {***e, s, F, G, H, C***} by 1. Similarly the Kernel Net - $\omega(R_u)$ - is the net obtained by replacing the same undetermined elements by zero.

Definition 4.4: Maximally (Minimally) Connected Organization: A maximal (minimal) element of the set of all feasible organizations, satisfying user-defined and structural constraints, will be called a Maximally (Minimally) Connected Organization, MAXO (MINO). The set of all MAXOs and the set of all MINOs will be denoted as $W_{max}(R)$ and $W_{min}(R)$, respectively.

Maximally and minimally connected organizations can be interpreted as follows. A MAXO is a net such that it is not possible to add a single link without violating the set of all constraints R, i.e., without crossing the boundaries of the subset W(R). Similarly, a MINO is a net such that it is not possible to remove a single link without violating the set of constraints R. Proposition 4.7 that follows is a direct consequence of the definition of maximal and minimal elements.

For the organizations with compound node DMUs, the solution set, the set of feasible organizations, can be completely characterized by its boundaries, the MAXOs and MINOs, due to the fact that all the constraints defined for compound node organizations are convex. In this case, constraints C1 and C2 imply R1. Therefore, in this case, an optional link is the incremental unit leading from a feasible net to its immediate super-ordinate. On the other hand, the set of feasible organizations with human DMs as DMUs may not be characterized by MAXOs and MINOs alone due to the non-convexity of constraint R1 (Remy and Levis, 1988). This problem was solved by Remy and Levis (1988) with the definition of Simple Paths (Sp). The constraint R1 is automatically satisfied, if a simple path of the Universal Net is taken as the building unit from one feasible net to another. Propositions 4.7 and 4.8 characterize the set of feasible organizations.

Definition 4.5: Simple Paths: Let Σ be a net that satisfies constraint R1 and whose source and sink have been merged together into a single external place. If the source and sink places of a Σ are merged together to form an external place, then a simple path of Σ is defined to be a directed elementary circuit which includes the external place.

Proposition 4.7: Let Σ be a net of a compound node of dimension m defined in a stratum k. Σ will be a feasible organization if and only if

- Σ is a union of simple paths of the Universal Net, i.e., $\Sigma \in \cup Sp(R_u)$.
- Σ is bounded by at least one MINO and one MAXO.
  $W(R) = \{\Sigma \in \cup Sp(R_u) \mid \exists\ (\Sigma_{min}, \Sigma_{max}) \in W_{min}(R) \times W_{max}(R)\ \Sigma_{min} \leq \Sigma \leq \Sigma_{max}\}$

Proposition 4.8: Let Σ be a net of a compound node of dimension m defined in a stratum k, where $k \neq N$. Σ will be a feasible organization if and only if

- Σ is bounded by at least one MINO and one MAXO.
  $W(R) = \{\Sigma \in W(R_u) \mid \exists\ (\Sigma_{min}, \Sigma_{max}) \in W_{min}(R) \times W_{max}(R)\ \Sigma_{min} \leq \Sigma \leq \Sigma_{max}\}$

Once the set of feasible organizations is characterized, one of them can be selected on the basis of some pre-defined performance criteria. In the example being illustrated, the computation of solutions resulted in the following:

- 4 Simple Paths, 1 MAXO and 1 MINO for Σ_{12}.
- 10 Simple Paths, 1 MAXO and 2 MINOs for Σ_{22}.
- 6 Simple Paths, 1 MAXO and 1 MINO for Σ_{32}.
- 4 Optional links, 1 MAXO and 2 MINOs for Σ_{11}.

The selected nets for all organizational structures are shown in Figs. 4.13, 4.14, 4.15, and 4.16.

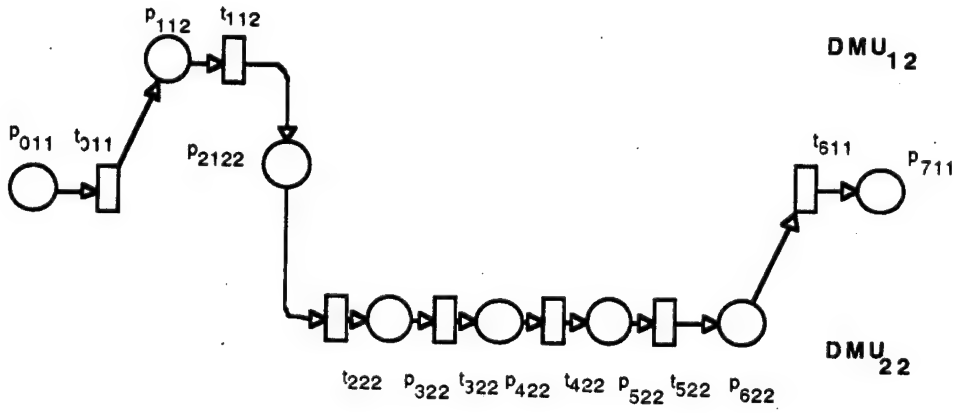


Figure 4.13 Selected net Σ_{12}

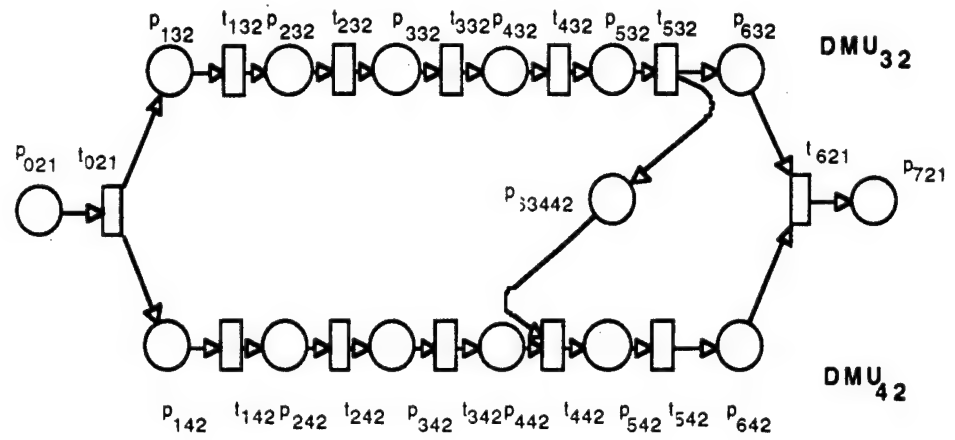


Figure 4.14 Selected net Σ_{22}

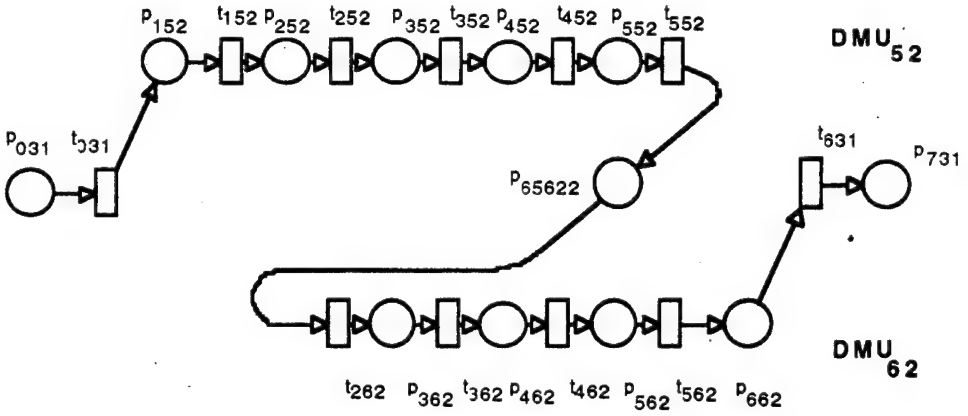


Figure 4.15 Selected net Σ_{32}

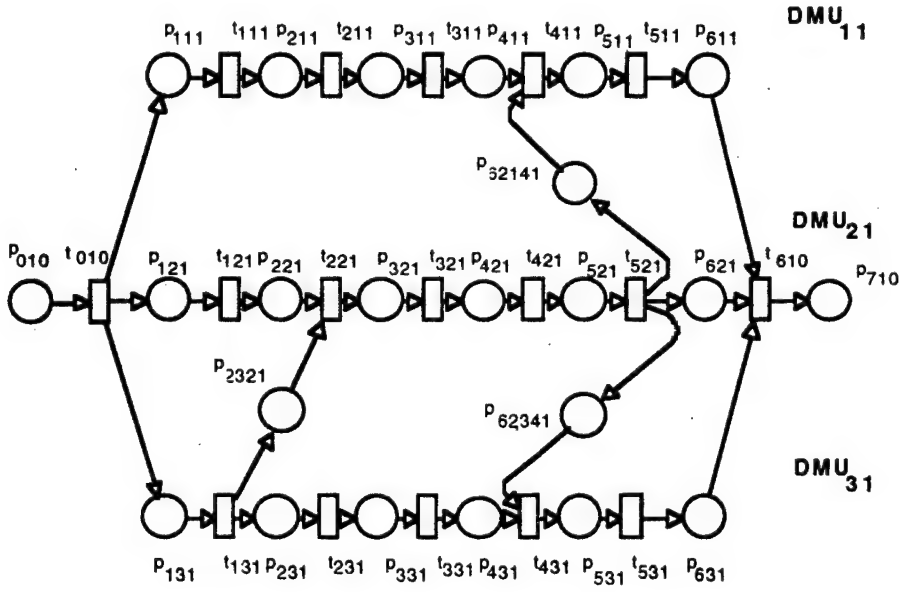


Figure 4.16 Selected net Σ_{11}

The labels shown in the figures refer to a labeling scheme used by Zaidi (1991) in order to keep track of the computations in the design algorithm. Figure 4.17 shows the organizations structure of DMU_{10} in terms of the DMUs defined in stratum 2. This more detailed description of the system, the stratum 2 description, is obtained by replacing the compound node representations in stratum 1 by their organizational structures as depicted by Figs. 4.13, 4.14, and 4.15. This constitutes the *unfolding* of the organizational structure from stratum 1 to stratum 2.

In Figure 4.17, all the arcs connected to places labeled as P_{62141}, P_{62341}, and P_{2321} (drawn shaded) show all the possible ways in which the interactions represented by these places in stratum 1 can be translated to their stratum 2 representations.

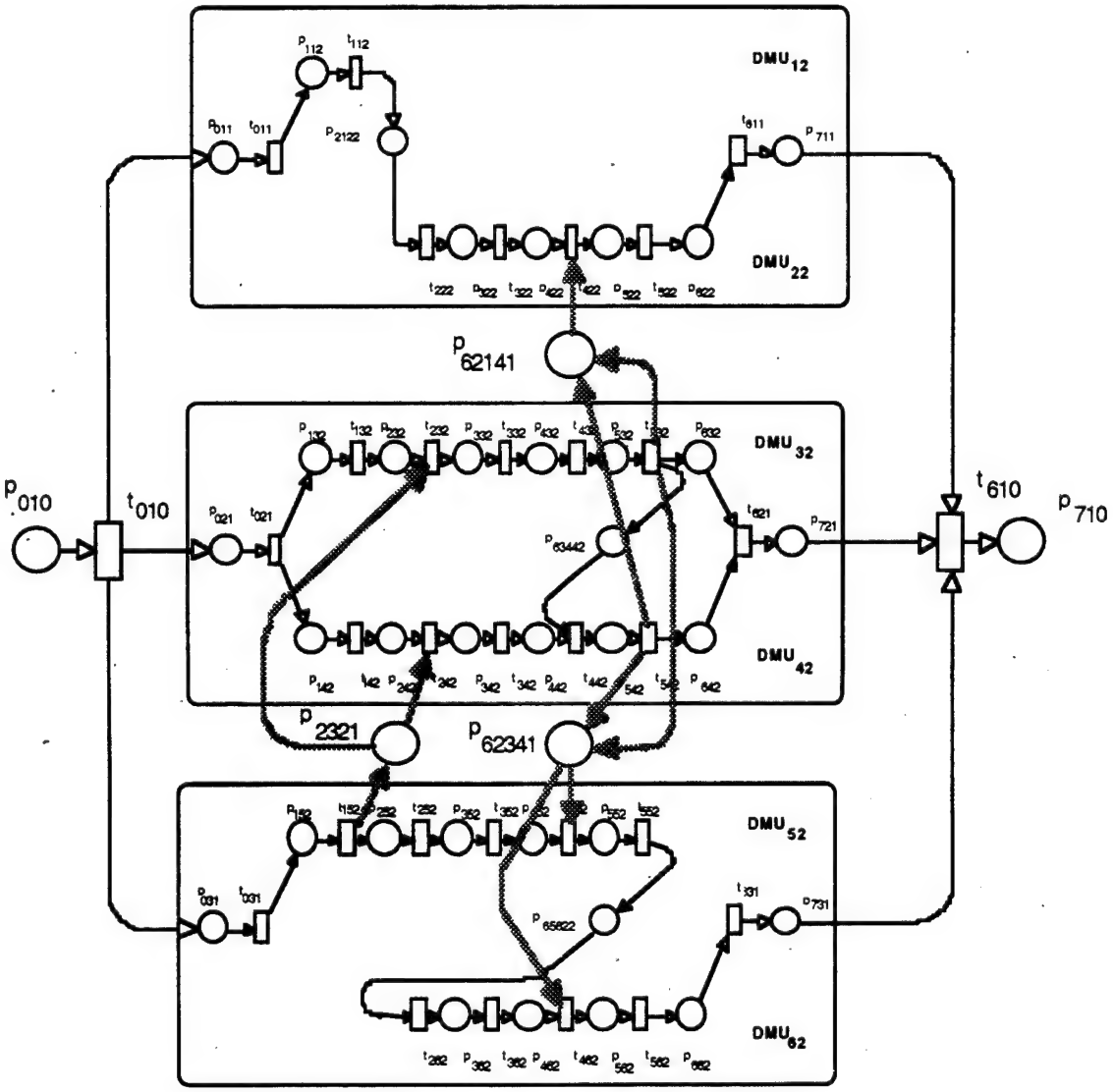


Figure 4.17 Stratum 2 Description of the System

4.2.8 Solution To Connectivity Problem

The problem of interpreting higher level interactions in terms of their lower level representation arises when an organizational structure is unfolded to its lower level description. A set of connectivity rules need to be formulated to resolve this connectivity problem. The connectivity rules presented in this section are based upon the multi-echelon hierarchical relationship that may exist among the DMUs of an organizational structure. *Echelons* refer to the mutual relationship among DMUs of an organizational structure; they define *super-ordinate* and *subordinate* DMUs within an organization. In order to define the multiechelon hierarchy among organizational members, the messages that flow in an organization are classified into the following categories:

1. *Information, INF*: Messages conveying information (INF) are further divided into three subcategories, inputs/outputs, assessments, and responses.
2. *Control Signals, CTR*
3. *Commands, CMD*

Inputs represent observations from the external environment (e.g., from sensors). They are modeled by the ***e*** array. Assessments are defined to be the outputs of the situation assessment stage of a DMU, modeled by ***F*** type interactions. The messages containing information about the response of a DMU are taken as responses, represented by ***H*** type links. The control signals contain, in addition to a limited amount of information about the task, an enabling signal for the initiation of a sub-task, as depicted by ***G*** type interactions. If the response or course of action selected by a DMU is dependent upon the message sent by another DMU, then such a message is termed a command or order; a ***C*** type interaction models such a situation. Therefore, the interactions of a DMU are divided into two classes: *input* interactions, and *output* interactions.

The three classifications of the organizational data yield $2^3-1 = 7$ different input/output interactional structures for a DMU. The seven possible ways in which a DMU can receive input or send output messages are given in the first column of Table 4.1.

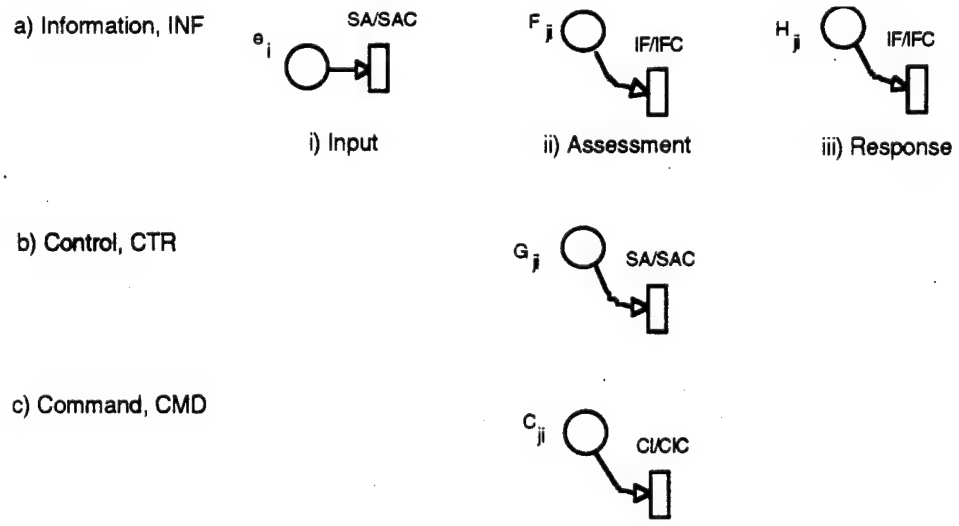
Let the outputs from a set of DMUs be taken as constant and let only the input interactions of the DMUs be considered. Then, a DMU receiving CTR or CMD type of messages is considered at a lower echelon than the one receiving INF messages. A number of sub-levels are also defined within the DMUs having INF as input interaction. The DMUs receiving responses are taken at a higher echelon than the DMUs receiving inputs or assessments. Similarly, DMUs with assessment type of input interactions are considered at a higher echelon than the DMUs with input type of INF.

Table 4-1 Ordering in Terms of Inputs and Outputs

Input Interactions	Corresponding Order on Inputs, I	Corresponding Order on Outputs, O
INF	1	7
INF, CTR	2	6
CTR	3	5
INF, CMD	4	4
INF, CTR, CMD	5	3
CTR, CMD	6	2
CMD	7	1

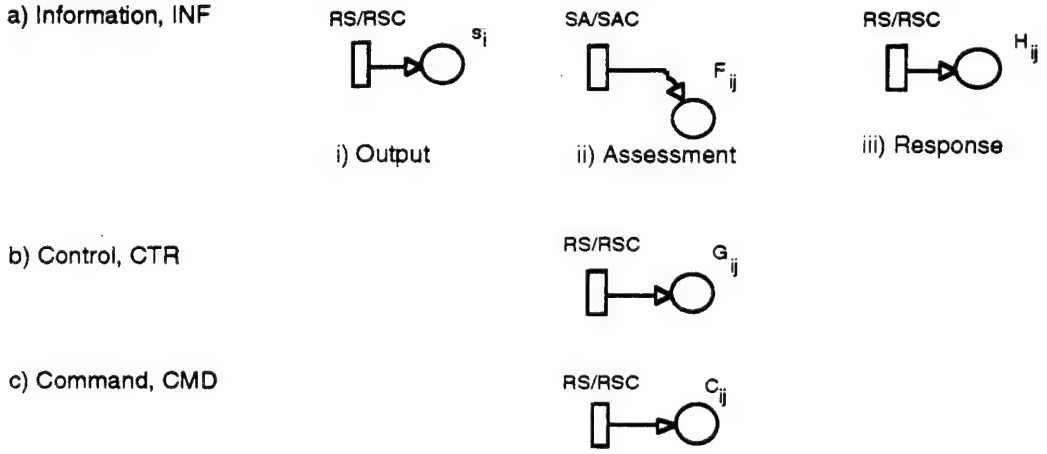
A DMU with CTR input is considered at a higher echelon than one with CMD inputs. The DMUs with all other combinations of input interactions fall within these three echelons. Column 2 of Table 4-1 shows the corresponding ordering for input interactions. A DMU with an order 1 is considered at the highest echelon as compared to all other DMUs with the identical set of output interactions. Column 3 of Table 4-1 presents the corresponding ordering for the DMU based on their echelon definition of the outputs.

Figure 4.18 shows a possible interpretation of the interactional links introduced in the previous section. The figure maps the classes of messages presented to their Petri Net representation in view of the physical interpretation of the interactional links.



(a) Classification of Input Interactions

Figure 4.18 Classification of Input and Output Interactions



(b) Classification of Output Interactions

Figure 4.18 Classification of Input and Output Interactions

An echelon index is defined for a DMU based on both input and output interactional structures of the DMU. A DMU is characterized as a 2-tuple, (I, O), where I corresponds to the order defined by the input interactions of the DMU, and O represents the order defined by the output interactions. The set of all the elements of the matrix is represented by $\prod$.

The lattice structure of $\prod$ is shown in Figure 4.19; it is the result of the partial ordering that exists between the elements of the set $\prod$. The arrows represent the relation "is higher than", i.e., A → B means that A is higher than B. The echelon index for a DMU is defined by the following equation.

After unfolding a compound node to the next lower stratum, each of the DMUs of the compound nodes is identified as one of the elements of the set $\prod$. Once the echelon indices associated with all the subsystems of the compound node are identified, a number of connectivity rules are applied to translate an interactional link defined in a higher stratum to its lower-stratum description.

Rule 1: An interactional link defined at stratum k from a compound node i to another compound node j is translated into a single link at stratum k+1 between the subsystems of the compound nodes i and j.

Rule 2: The translated lower stratum interactional link between the subsystems of the compound nodes i and j will connect the highest echelon-DMUs of the two sub-organizational structures. The highest echelons identified for the subsystems of i and j need not necessarily be the same.

Rule 3: If a compound node has two or more DMUs at the same highest echelon, the following rule applies:

- For an output interaction the DMU with higher O index is selected.
- For an input interaction the DMU with higher I index is selected.
- For two or more DMUs with identical (I, O) indices, one of them is selected arbitrarily.

Rule 4: If, in following Rules 1 to 3, constraint R1 or R2 is violated, then the next highest echelon-DMU will be selected to participate in the interaction. The identification of the next highest echelon-DMU follows the procedure presented in Rules 2 and 3.

Echelon Index = I + O - 2

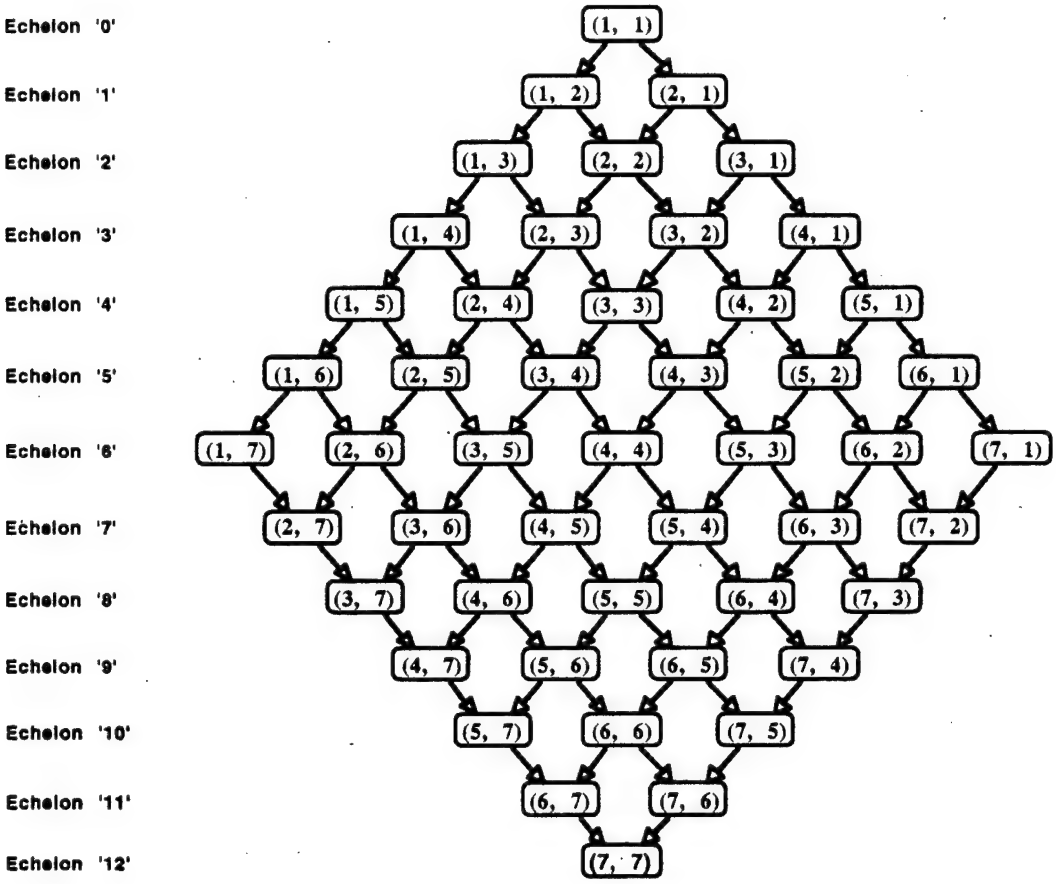


Figure 4.19 Multi-echelon hierarchy

The rules stated above are applied to the organizational structure in Figure 4.17. As a result of the identification of echelon indices for the DMUs in stratum 2 and the application of the connectivity rules, the arcs connected to the places, P_{62141}, P_{62341}, and P_{2321}, drawn by solid lines in Figure

4.20, represent the lower level connectivity of the higher level connections. The organizational structure shown in Figure 4.20 represents the system's description at the lowest stratum.

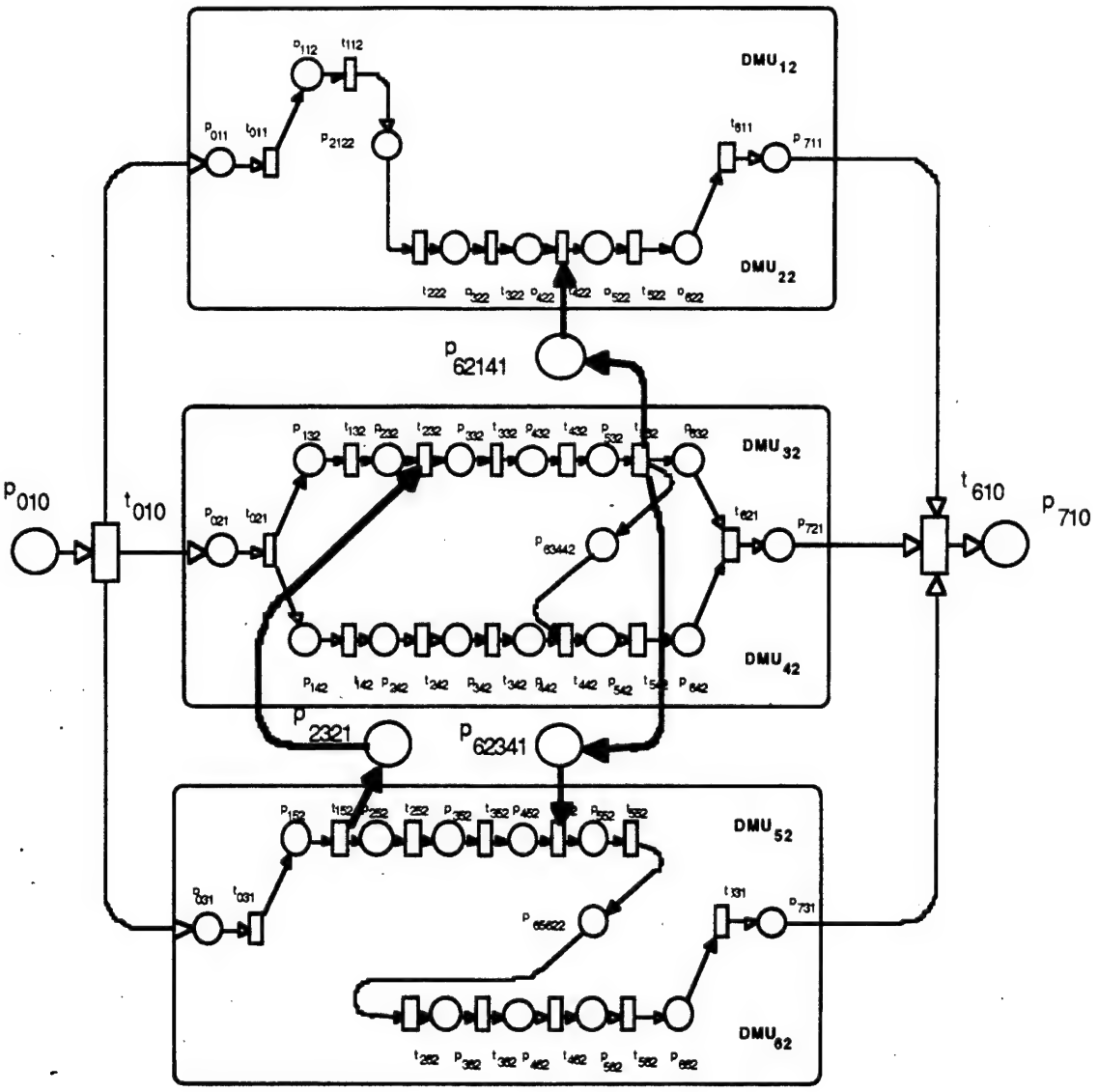


Figure 4.20 Startum 2 Description of the System

4.2.9 Conclusions

An algorithmic design of the multilevel organizations has been presented. The methodology provides a natural, structured, and modular way for formulating and solving the problem of designing the organizational structure of a distributed inteligence system. An organization with hundreds of lower level subsystems can be modeled with less computational effort by carefully

defining the higher level subsystems of the organization in terms of the lower level ones. The entire organization can then be modeled only in terms of the higher level subsystems. Finally, all the structures are integrated to produce a family of structures for the organization each describing the organization at different level of detail.

In the next section, coordination in organizations is addressed and the concept of the coordination constraint, already mentioned here, is explored.

4.3 COORDINATION IN DISTRIBUTED DECISION MAKING ORGANIZATIONS

The need for Distributed Decision Making Organizations (DMOs) emerges when the amount of processing work exceeds the processing capacity of one single intelligent node. A task has to be partitioned into sub-tasks and distributed to different sub-systems; coordination among sub-systems becomes necessary. A distributed decision making organization consists of intelligent nodes, where each intelligent node carries out certain functions and interacts with other nodes. A processing task is distributed to different nodes. Therefore, the entire system can be viewed as an interconnection of sub-systems, with each sub-system performing part of the overall task. Each sub-system, if well coordinated, contributes to the performance of the entire organization.

The concept of an organization embodies two meanings. One is the set of physical entities and the interactions between them which form the physical part of an organization. Another is the set of rules that govern the interactions among these entities. We call all these physical entities and their interactions the system, and we characterize the internal operation of the system as coordination.

These two concepts could be successfully de-coupled at the modeling stage using Colored Petri Nets (Lu, 1992). The organizational model consists of two layers, the System Layer and the Coordination Layer. The System Layer models all the physical entities (including their interactions) and the processes they are designed to perform; and the Coordination Layer models the rules of operation that each physical entity must follow if it is to be part of the organization.

An organization consists of components and their interactions. A *component* of an organization is defined to be a physical entity with its operating rules. It can perform a set of basic tasks that is relevant to the context in which the organization operates. Figure 4.21 describes an organization consisting of four components C1, C2, C3 and C4. Each of them performs a certain function in a certain scenario. Input1, Input2 and Input3 are its inputs; Output1, Output2 and Output3 are its outputs.

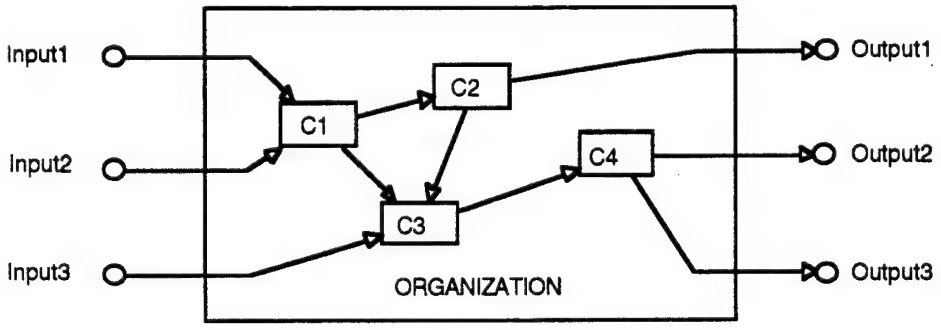


Figure 4.21 An Organization of Four Components

An organization has a variable structure, if the interactions between the components that belong to the organization and the process each component performs can vary in response to external or internal changes.

Monguillet and Levis (1993) initiated the investigation of variable structure Decision Making Organizations. Based on the theory of Predicate Transition Nets (PTN) (Genrich, 1987) the framework of System Effectiveness Analysis was extended to compare both variable and fixed organizations. An appropriate mathematical framework, that of Colored Petri Nets (CPN) (Jensen, 1992) was defined to investigate systems that adapt their structure of interactions to the input they process (Demaël and Levis, 1994).

Coordination defines the way a system operates. It describes how each component behaves under a certain scenario (e.g. what task it should perform, for what type of inputs it should wait and to what components it should send the outputs.). The *Coordination Layer* of an organization gives the description of the operation rules for all components in an organization.

One sub-problem in designing an organization is designing the Coordination Layer given the System Layer. One objective for doing this is to specify rules on how to vary the interconnections of each component with other components. To obtain these rules, a Coordination Constraint must be considered. The nature and form of this constraint is one of the most important issues in designing variable structure organizations.

This section of the report first presents an introduction to a modeling approach for constructing organizational structures. Then, the Coordination Constraint is addressed in detail and an algorithm for checking the feasibility of the coordination structure of an organization is presented. A mathematical framework, based on Colored Petri Nets, is developed for representing variable structure organizations.

4.3.1 Colored Petri Net Modeling of an Organization

A *Coordination Strategy* is defined as a function from the inputs of a component to its responses. For example, in Figure 4.22, T models a component. Let color set A = {a1, a2} be associated with place p1, B = {b1, b2} be associated with place p2, and C = {red, blue} be associated with p3 and p4. The coordination strategy can be expressed by a set of rules as follows:

If there is a token of type a1 in p1 and no token in p2, firing T results in a blue token in p3.

If there is a token of type a1 in p1 and a token of type b1 in p2, firing T results in a red token in p4.

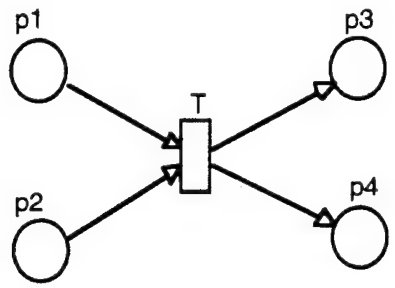


Figure 4.22 A Component with Two Input and Two Output Places

All possible input patterns of the component need to be listed (The list should be exhaustive). For any possible input pattern, there exists one and only one response corresponding to that input pattern. This relation of responses and input patterns is a function defined on the set of input patterns. This is a Coordination Strategy. Design of a Coordination Strategy is to determine the set of all possible input patterns, and identify the corresponding response for each input pattern.

Modeling of both the Coordination Strategies and tasks is realized using the Colored Petri Net formalism. The remaining part of this section shows how Coordination Strategies can be modeled explicitly and be separated from the tasks.

Status Place

Consider the example in Figure 4.22, and assume that the following rule is in the list: if there is one a1 token in p1 and one b2 token in p2, then fire T to remove the a1 token from p1 and put a red token in p3. The Colored Petri Net as in Figure 4.22 cannot realize this rule, because if the enablement condition is defined as one a1 in p1 and one b2 in p2, the firing of T will remove also the b2 token in p2. The information that there is a b2 token in p2 is used only for affecting the firing result of transition T, but the token itself should not participate in the firing. A status place is introduced to implement such firing rules for a transition. The concept of a status place also leads to the de-coupling of an organization into two layers.

In Colored Petri Net theory, a place can hold multiple tokens in different classes. The *status of a place* contains information on the multiset of tokens in that place. For example, if there are two red,

"r", and three blue, "b", tokens in place P, then the status of place P is "2r + 3b". The idea of a status place is to fold the information on the marking of a place into the color of a single token.

Two situations need be considered when implementing the concept of status place. In the first case, the number of token classes the place can hold is limited, but the total number of tokens the place can hold has no limit. In the second case, the total number of tokens the place can hold is limited (the place has capacity limits). The ways of implementing a status place are different for the two cases.

First Case

In this case, the number of token classes the place can hold is limited. For example, place P has one input and one output transition (Figure 4.23) and it can hold m classes of tokens represented as Class 1...Class m. We are going to design a structure so that the status of place P can be obtained.

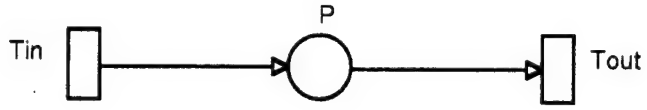


Figure 4.23 A Place with One Input and One Output Transition

First we add the place S on the net which is called the *status place* of place P. It is a place which links the input transition (Tin) to place P with a bi-directional arc or two directed arcs, one from S to Tin and one from Tin to S (Figure 4.24). Similarly, a bi-directional arc links Tout to S. A token carrying an m-dimensional vector called the *status token* of place P always resides inside the place S; the vector can be represented by $(n_1, n_2, .., n_i, .. n_m)$.

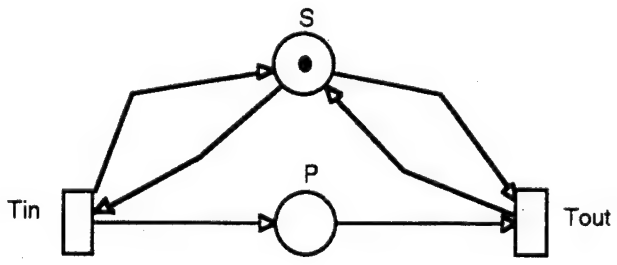


Figure 4.24 Place P with its Status Place S

Methodology for realization:

Initialize the status token in S so that

n_1 = the number of tokens of $Class_1$ in P,

n_2 = the number of tokens of $Class_2$ in P,

... ...

n_i = the number of tokens of $Class_i$ in P,

... ...

n_m = the number of tokens of $Class_m$ in P.

Set up a rule for Tin and Tout so that whenever Tin generates tokens in P or Tout withdraws tokens from place P, the magnitude of each n_i in (n_1, n_2, .., n_i,.. n_m) is changed so that it is always equal to the remaining number of tokens of Class i in place P.

Example:

Place P can hold two classes of tokens: red and blue. The firing rule for Tin places one red and two blue tokens in place P; the firing rule for Tout removes one red and one blue token from place P. The initial marking of place P is null.

Let us construct a status place S and its arcs as in Figure 4.25. There is a token in S with a two-dimensional attribute vector represented by (n_1, n_2). The first component corresponds to the number of red tokens in place P; the second one corresponds to the number of blue tokens. The initial marking of S is a token with color (0, 0). Assume that, before firing, the marking of S is (n_1, n_2). Let the firing rule for Tin be that the marking of S becomes (n_1 + 1, n_2 + 2) after firing. Let the firing rule for Tout be that the marking of S becomes (n_1 - 1, n_2 - 1) after firing. For example, the marking in place P is null, and the color of the token in S is (0, 0). At this time Tin fires and the marking of P becomes one red and two blue; the token in S changes to (1, 2). Then Tout fires and the marking of P becomes one blue; the token in S changes to (0, 1). The color of the token in S indicates that there is one blue token in P. Therefore, by reading the status token in place S, we know the number of tokens of different classes in place P (the status of place P).

Figure 4.25 shows the general case for place P when there are j input and k output transitions. There is always a token residing in S with attribute vector components always equal to the number of tokens of each different class in place P.

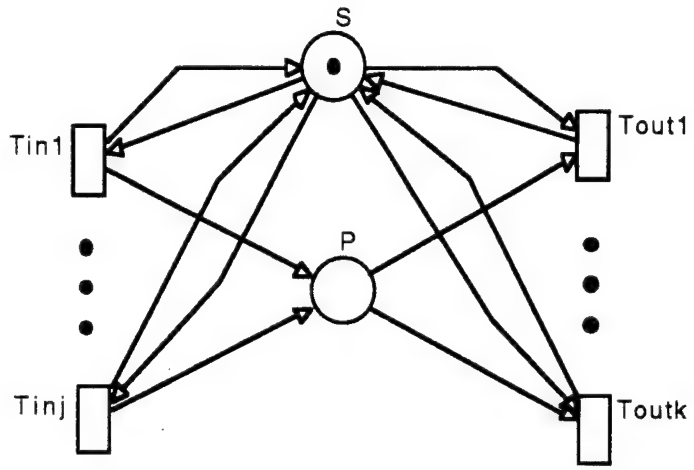


Figure 4.25 Status Place for Multiple Input and Output Transitions

Second Case

In this case, the total number of tokens place P can hold is limited, but there are no constraints on the classes of the tokens.

The methodology for implementing this status place is different from that of the first case. However, the structure of the status place is the same as in Figure 4.23. The implementation methodology is described as follows. In Figure 4.23, suppose place P can hold a maximum of m tokens and the initial marking is null. Let us construct a status place S as shown in Figure 4.24 with a status token of m dimensions (x_1, x_2, ..., x_m) residing in it. The initial marking of S is a token with color (null, null, ..., null). Make the firing rule for Tin and Tout to be: When a token with color id enters P, scan the vector components of the status token from left to right and the first null component found is changed from null to id (the color of that token is remembered by the status token). When a token with color id is removed from P, scan the vector components of the status token from right to left and the first component whose entry is id is changed back to null. It needs to be stated that the rule for scanning the vector is not important and will not affect the results.

Example:

Suppose place P can hold a maximum of three tokens, and that the initial marking of P is empty. Construct a place S as shown in Figure 4.24. At initialization, put a token in S with color (null, null, null). When Tin fires, a blue token is generated in P, and the status token changes to (blue, null, null). When Tin fires again, it generates a red and a blue token in P; the status token changes to (blue, red, blue). When Tout fires, a blue token is removed from P; the status token changes to (blue, red, null). By reading the color of the status token, information on tokens in place P is obtained.

Colored Petri Net Modeling of Coordination Strategies

Suppose we need to implement a Coordination Strategy for the transition in Figure 4.22. In order to separate the Coordination Strategy and the tasks, we implement place s1 as status place for p1 and s2 as status place for p2 and modify slightly the structure of the status place. Figure 4.26 shows how the tasks and the Coordination Strategy can be separated. As shown in Figure 4.26, we link s1 and s2 by bi-directional arcs to another transition C instead of transition T. Therefore, Coordination Strategies are implemented in transition C and tasks are implemented in transition T. The following example shows how to implement a Coordination Strategy.

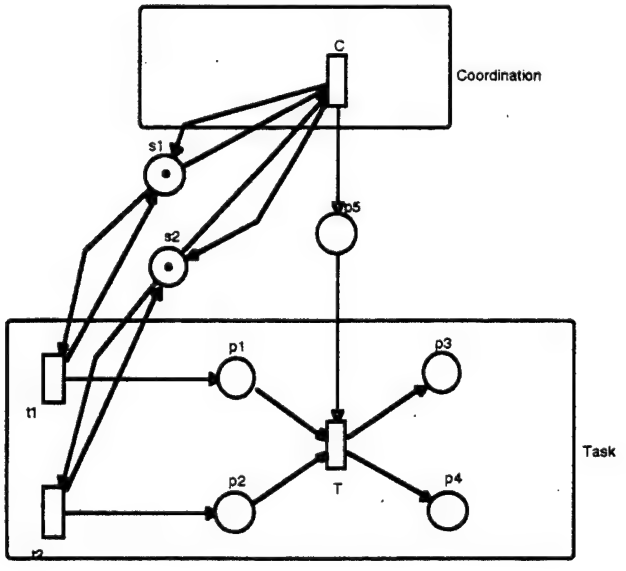


Figure 4.26 Separating the Task and the Coordination Strategy

Example:

In Figure 4.27, suppose that place P can hold three classes of tokens: red, blue and green. The following firing priority rule for Tout must be implemented: Priority [green] > Priority [blue] > Priority [red] or if there is any green token in place p then fire the green token; if there is no green token but there is a blue token in place p, then fire the blue token; if there are no green or blue tokens but a red token, then fire the red token.

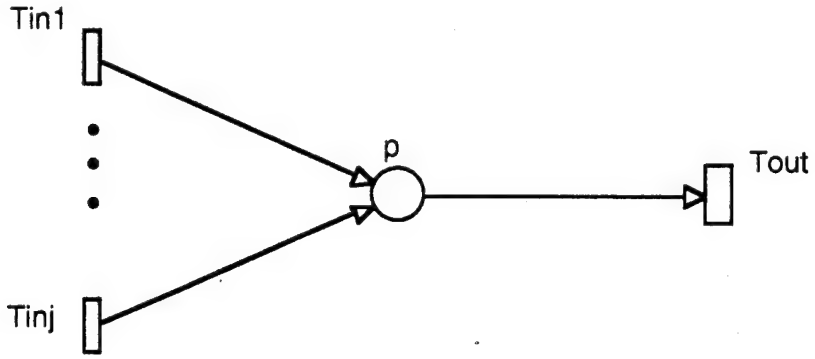


Figure 4.27 Priority for Place p to be Implemented

Let us construct a status place S (Figure 4.28) with a status token in it whose color is: (n_1, n_2, n_3). The first component corresponds to the number of red tokens, the second to the number of blue tokens, and the third to the number of green tokens. The priority firing rule for T_c is:

- if n3 > 0 put a control token in place c which will fire a green token in P
- else, if n2 > 0 put a control token in place c which will fire a blue token in P
- else, if n1 > 0 put a control token in place c which will fire a red token in P.

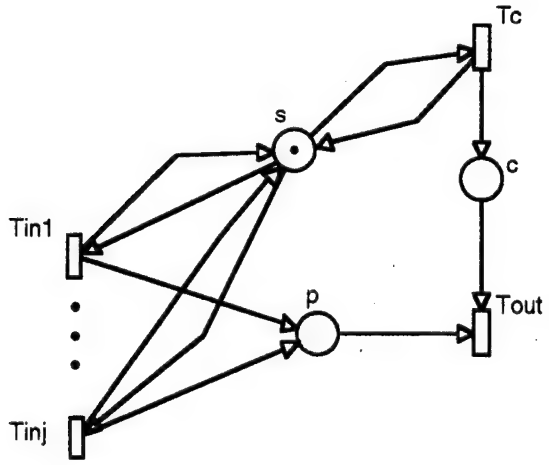


Figure 4.28 Implementation of Priority as a Coordination Strategy

For example, if there is one blue and two red tokens in place P, transition Tc checks n3 first and finds that it is not larger than zero, then it checks n2 and finds it is larger than zero. It then puts a control token in place c to fire with a blue token in place P; the priority rule has been realized.

The Two Layered Representation of a Component: Modeling a Component

As we stated earlier, the organization consists of components with two layers. Figure 4.29 explains the relationship between the two layers. The System Layer models the system. It depicts the tasks each component can perform and the interactions among components. The inputs (material/information) to the system will follow some paths in the System Layer and produce outputs.

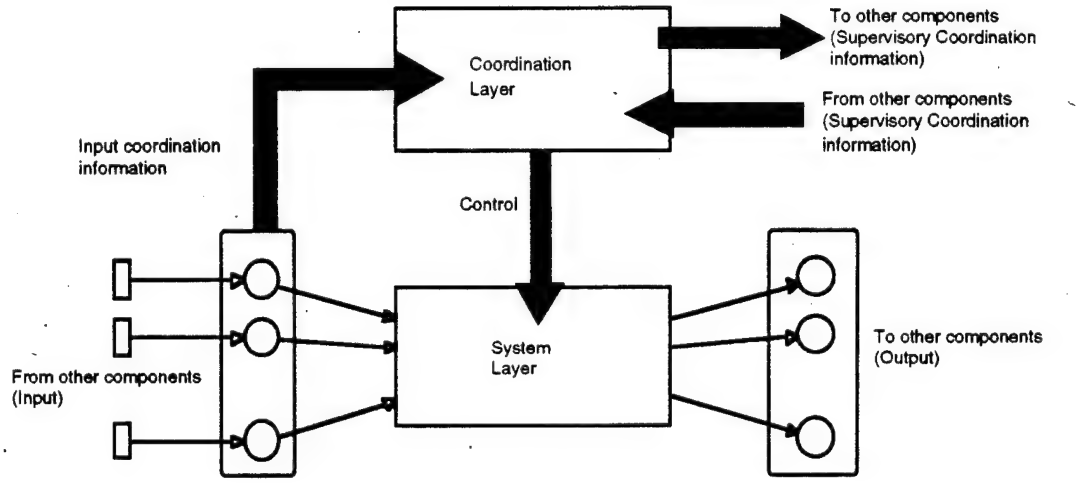


Figure 4.29 A Component with Two Layers

The Coordination Layer of a component depicts the rules which govern the operation of this component. It also has interactions with the Coordination Layer of other components. Therefore, supervisory coordination information can be passed between components. This kind of information flow paths is different from the one in the System Layer. Information flows in the Coordination Layer are related only to coordination (the changes of the environment and system parameters, etc.).

Input information also contains coordination information, This part of the information will flow to the Coordination Layer. According to a given set of coordination rules, the Coordination Layer will generate controls to the system to govern the behavior of the component. Because the input

information is local to the component, only the status information in the input places of the component can flow to the Coordination Layer of this component.

Modeling the System Layer of a Component

Figure 4.30 shows the System Layer of a component. P1, P2, ..., Pp are input places that represent interconnections with the external environment or other components. The transitions f1, f2, ..., fn model the alternative tasks the component can perform. The places O1, O2, ..., Oq are output places through which the component will send the result to other components or to the external environment. c1 and c2 are the two control places which control the behavior of the component, they act as the interface to the Coordination Layer.

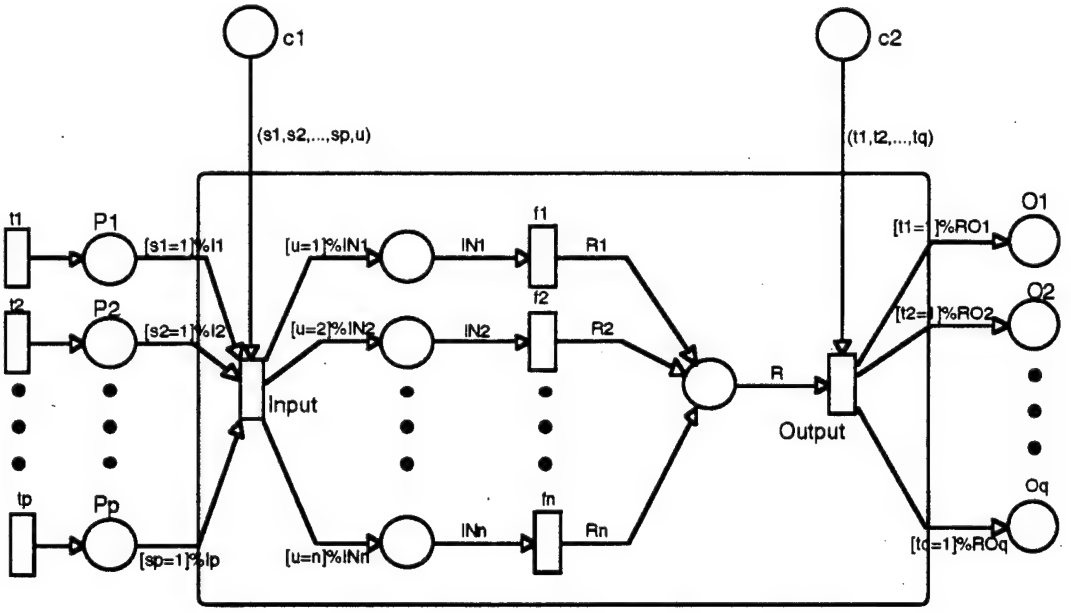


Figure 4.30 The System Layer of a Component

The System Layer of a component consists of a set of transitions **FS** = {f1, f2, ..., fn} which will be executed in different processes. The interactions to and from other components are represented by **ES** = {places P1, P2, ..., Pp and their output arcs to transition Input, and places O1, O2, ..., Oq and their input arcs from transition Output}.

The Coordination Layer has to determine what kind of inputs the components are expecting, which task should be chosen and to which component/s it should send the output. Places c1 and c2 are the control places from the Coordination Layer. A token in c1 with a vector (s_1, s_2, ..., s_p, u) can control the input to the component and the task to be performed. A token with a vector (t_1, t_2, ...,

t_q) can control the output of the component. For example, if a token in c1 is (1, 0, ..., 0, 2), the token in P1 will be used and function f2 will be chosen. If a token in c2 is (0, 1, 0, ..., 0, 1), the output of the component will be sent to O2 and Oq.

The color in the input places of a component is a tuple, which can be partitioned into two parts, $I = I_C + I_i$, where I is the tuple of the token, I_C denotes the components of the tuple which relate to coordination, while I_i denotes the components of the tuple which do not relate to coordination. Only I_C will be sent to the Coordination Layer.

Modeling The Coordination Layer of a Component

In the System Layer of a component, a function CS can be defined on the set of all possible input patterns. For each kind of input pattern, there is a response corresponding to it. The rules for the function constitute a Coordination Strategy for this component.

Example:

Figure 4.31 shows a component. Suppose P1 has a color set A={a1, a2} and P2 has a color set B={b1, b2}. We define the rules as follows:

- If there is an a1 in P1 and a b1 in P2, then select f1, and generate an output token to O1 only.
- If there is an a2 in P1, regardless of the marking of P2, then select f2, and generate output tokens in both output places O1 and O2.
- If ..., then ....

All possible input patterns need to be defined in the list. No two rules should be effective concurrently. Because the rules are defined for **all possible input patterns**, the input pattern which can enable two rules should be also defined as a rule in the list. After the input patterns have been defined, a Coordination Strategy or even a set of alternative Coordination Strategies can be defined for this component.

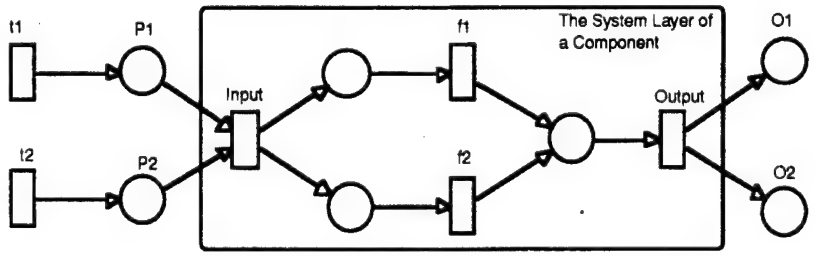


Figure 4.31 A Component with Two Alternative Tasks

The Coordination Layer of a component may have a set of Coordination Strategies. Figure 4.32 models the Coordination Layer using Colored Petri Nets. In the figure, the System Layer of the component is represented by a single transition and has p input places from the System Layer of other components and q output places to the System Layer of other components. Places P1', P2', ..., Pp' are the status places for input places P1, P2, ..., Pp.

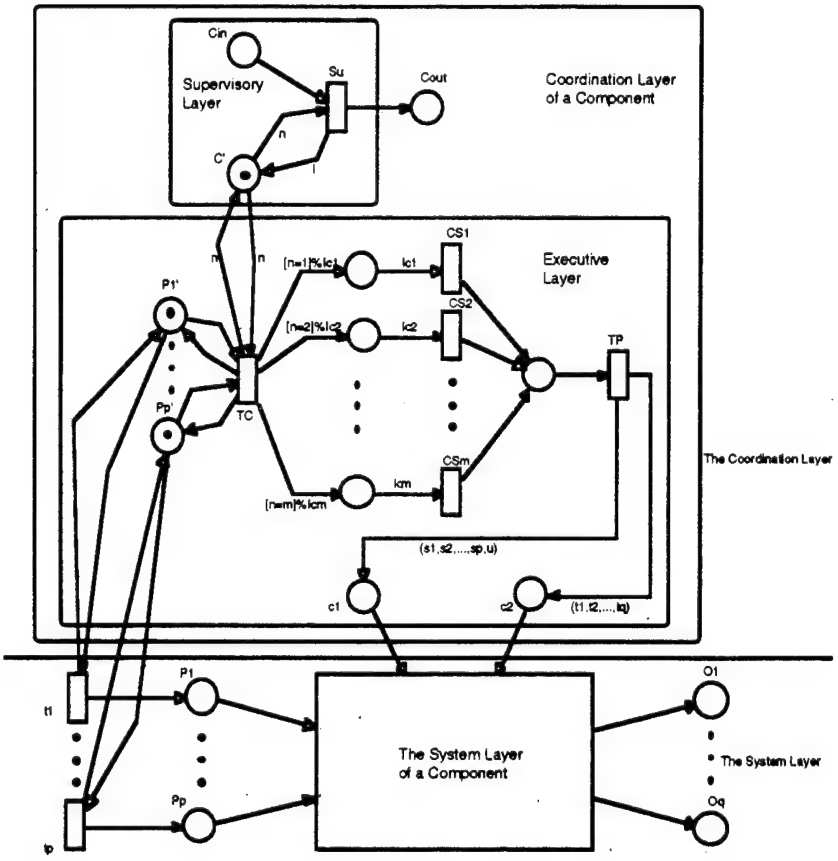


Figure 4.32 The Coordination Layer of a Component

The Coordination Layer can be sub-divided into two layers: the Supervisory Layer and the Executive Layer, based on the different roles they play in coordination.

In Figure 4.32, the Executive Layer of a component consists of a set of Coordination Strategies **CSS**={Coordination Strategies in CS1, CS2, ...,CSm}, where CS1, CS2, ..., CSm are transitions with Coordination Strategies embedded in them. Each transition CSi (i = 1, ..., m) represents one Coordination Strategy. The rules are implemented in these transitions. The information that comes from the System Layer is denoted by **SC**={The information stored in the

status tokens in status places P1', P2', ..., Pp'.}, and the controls to the System Layer **CtrlS**={Control Place c1, c2}.

The Supervisory Layer consists of a transition Su and the set of rules **CR** = {rules embeded in Su} used to select the Coordination Strategy to be executed, the place C' which contains the controls to the Executive Layer **CtrlE** = {Place C' with the token in it} and the places Cin and Cout which contains the exchange of supervisory coordination information between different components **EC** = {Place Cin, Cout, Arcs to and from Su}.

Example:

If the color of the token in place C' is 2 (Figure 4.32), then when transition TC fires, it will choose to go to the second branch and go through transition CS2. Therefore, the Coordination Strategy CS2 is used.

Transition Su is responsible for changing the color of the token in place C' to control the selection of Coordination Strategy in the Executive Layer. The rules for switching Coordination Strategies are built in transition Su. Change of Coordination Strategy may take place, when an exchange of supervisory coordination information takes place. The enablement condition in transition TC determines the circumstances under which the component needs to respond to the input. Therefore, a component can be represented by the set {**FS, ES, CSS, SC, CtrlS, CR, CtrlE, EC**} where

- **FS** is a set of tasks in the System Layer of a component denoted as {f1, f2, ..., fn}.
- **ES** are the interactions between components in the System Layer denoted as the set {place P1, P2, ..., Pm and their output arcs to transition Input, place O1, O2, ...Ol and their input arcs from transition Output}.
- **CSS** is a set of Coordination Strategies in the Executive Layer of a component denoted as {CS1, CS2, ..., CSm}.

- **SC** is the coordination information that comes from the System Layer to the Executive Layer denoted as the set {Status tokens in places P1', P2', ..., Pp'}.
- **CtrlS** are the controls from the Executive Layer to the System Layer denoted as {Control Place c1, c2}.
- **CR** are the rules for selecting a Coordination Strategy denoted as {rules embeded in Su}.

- **CtrlE** is the control from the Supervisory Layer to the Executive Layer denoted as {Place C' with the token in it}.
- **EC** is the exchange of supervisory coordination information between the components denoted as {Place Cin, Cout, Arcs to and from Su}.

The Two Layered Representation of an Organization

The two layered representation decouples the coordination issues and the task issues in the same organization. Modifying the Coordination Layer can be accomplished without affecting the System Layer. This is important for both implementation and analysis.

Coordination Schemes

During the operation, every component of the organization is associated with a Coordination Strategy (called active Coordination Strategy), so that the organization can perform a well defined task. A Coordination Scheme is the set of active Coordination Strategies under which the organization is performing the task.

A Coordination Strategy and a Coordination Scheme are different. A Coordination Strategy is with respect to a component (a function from the input of the component to the response of that component). A Coordination Scheme is a set of active Coordination Strategies for all components with each component having one active Coordination Strategy.

An organization may have several schemes for doing the same task under different environment and system changes. One scheme can change to another scheme by changing the Coordination Strategies of the components. This can be accomplished by exchanging supervisory coordination information.

Variability and the Two Layered Representation

In the System Layer, all the tasks and interactions between components are implemented. Selection of tasks and the interactions between components is done by the Executive Layer. In the Executive Layer, there exists a collection of Coordination Strategies, but the selection of a Coordination Strategy is left to the Supervisory Layer. This layer determines the active Coordination Strategy according to the supervisory information provided by other components.

Different types of variability can be dealt with by different layers of the organization. The rules for the first type of variability (adaptability to inputs of the organization) are implemented in the Executive Layer, which gets information from the input places of the System Layer to generate controls according to certain rules. The rules for the second (adaptability to environment changes) and the third type of variability (adaptability to the system changes) can be built into the Supervisory Layer, which changes the Coordination Strategy according to the information about the system and the environment.

There are two interesting cases:

- If there is no variability due to input. The organization will act like a fixed structure (or in fixed mode) under normal situations. When there are changes in the organization or in the environment, the system will switch to another fixed structure (or fixed mode).
- If there is no variability due to environmental changes and system changes, the system will act as the model proposed by Demaël (1989) (variable structure or adaptation to the inputs).

4.3.2 Mathematical Model

In the previous section, a model of the System Layer and the Coordination Layer have been presented. This section presents a mathematical model for describing an organizational structure.

An organization processes data from n sources of information, i.e. sensors. Each sensor i (denoted as Si), i = [1, n], can output one letter from its associated alphabet $I_i = \{I_{i1}, I_{i2}, ..., I_{iai}\}$. These alphabets describe the basic items of information. The alphabets can include the null element, i.e., the case where no item of information is transmitted. A supersource can be created, which generates events (Stabile and Levis, 1984). In Figure 4.33, place I models the supersource, whose color set is $I = I_1 * I_2 * ... * I_n$. Transition T1 distributes the information sources from the super source to the appropriate places Ii, i = [1, n]; Ii is the color set for place Ii. Transitions Sensor 1, Sensor 2, ..., Sensor n model the n sensors of the organization, which detect information sources

$I_1, I_2, ..., I_n$, respectively. Place Output 1, Output 2, ..., Output l model the outputs of the organization. They converge through transition T2 into place Sink.

Definition 4.6: Suppose $I_1, I_2, ..., I_n$ are input color sets for the System Layer. An input token to the System Layer is represented by an n-dimensional vector $v = (v_1, v_2, ..., v_n)$ where $v_1 \in I_1$, $v_2 \in I_2, ..., v_n \in I_n$, i.e. $v \in I_1 * I_2 * ... * I_n$. Therefore, the cross product of the input color sets forms an n-dimensional input space. In the input space, there is a subspace which includes all feasible input vectors. We call this subspace *Feasible Input Domain I* or *Feasible Input Space*. An

element in the Feasible Input Domain, is called an *Input Situation* $v \in I$. *A token in the System Layer will not change its color when it moves through the organization.*

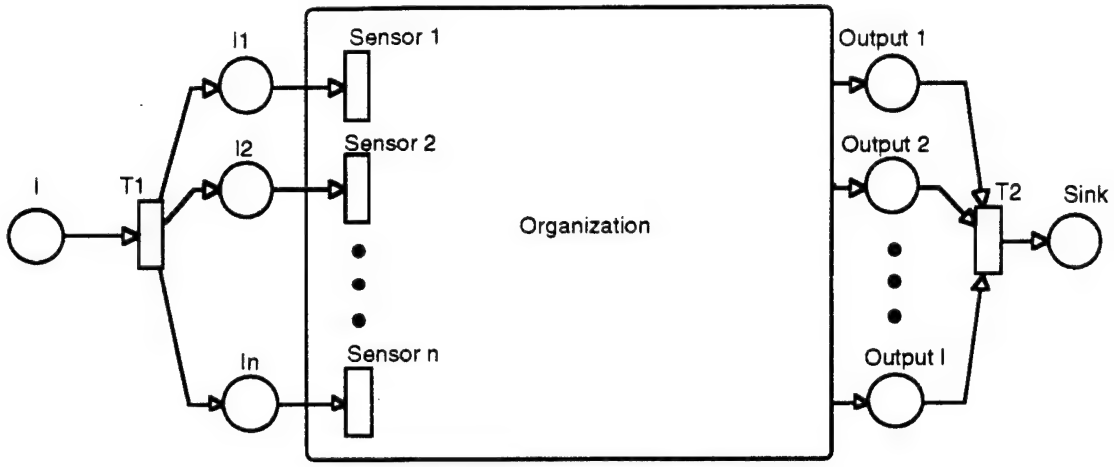


Figure 4.33 Information Sources and Sensors of an Organization

Assumption 4.1: The input color sets are assumed to be finite and discrete.

Definition 4.7: The System Layer is a Colored Petri Net representing all the components and their interactions. Each component is represented by a single transition. Places and arcs represent connectivity between components.

Here, the changes of system parameters are regarded as inputs to an organization. They are special inputs in the sense that they come from within the system itself. However, the inputs to the System Layer and the environment changes come from outside the organization.

Assumption 4.2: If the supersource place and the sink place are merged together into a single place, then the nets are marked graphs.

Fixed Structures and Variable Structures

Given the System Layer of an organization, and a token with color v, $v \in I$, the paths which the token follows through the System Layer and the task each component performs forms a fixed structure. An example is used here to illustrate the two concepts.

Figure 4.34 shows the System Layer of an organization. The shaded box indicates the boundaries of the organization. C1, C2, C3 and C4 are four components. Suppose there is one task in each

component (task f1 for C1, task f2 for C2, task f3 for C3, task f4 for C4.). When there is no input to a component, it remains idle. Suppose the color set for place I is I = A * B, where A = {a1, a2} and B = {b1, b2}.

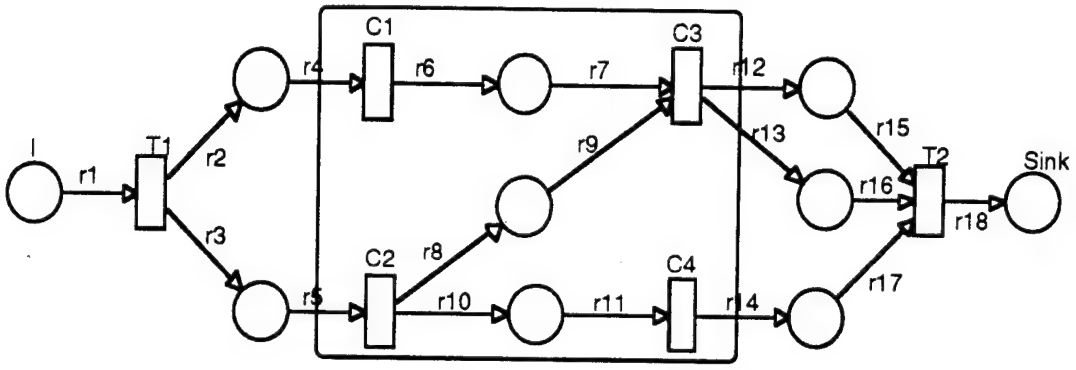


Figure 4.34 The System Layer of an Organization

Figure 4.35 shows a fixed structure for Input Situation (a1, b1). When a token with color (a1, b1) is put in place I, it follows the bold path through the organization; C1 does task f1, C2 does task f2, C3 does task f3, and C4 remains idle. The shaded path, while it exists physically, is not utilized when a token with color (a1, b1) passes through. Let us denote this fixed structure as (FS1). For tokens with color (a1, b2), there is also a fixed structure corresponding to it, denoted as (FS2) (Figure 4.36). In this case, C1 does task f1, C2 does task f2, C3 does task f3, and C4 does f4. Similarly, for Input Situation (a2, b1), there is a fixed structure corresponding to it, denoted as (FS3) (Figure 4.37). In this case, C1 does task f1, C2 does task f2, C3 does task f3, and C4 remains idle. For Input Situation (a2, b2), the fixed structure corresponding to it is the same as that of (a1, b1) denoted as (FS1) (Figure 4.35).

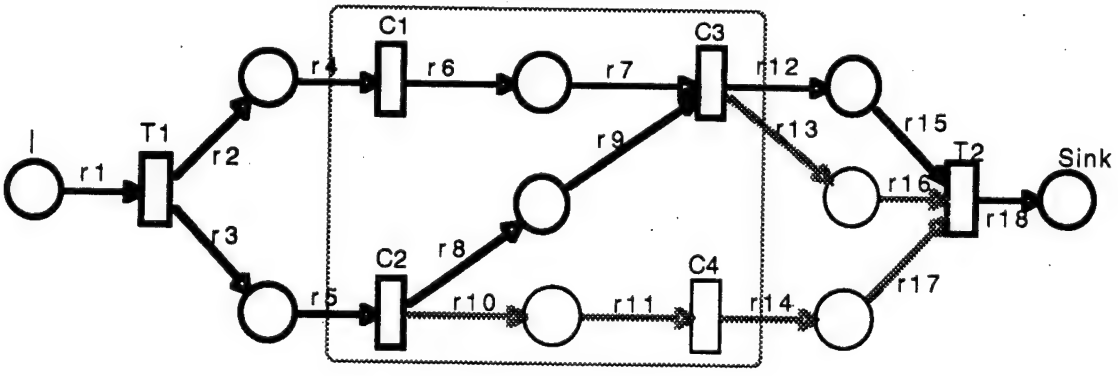


Figure 4.35 Fixed Structure FS1 for (a1, b1)

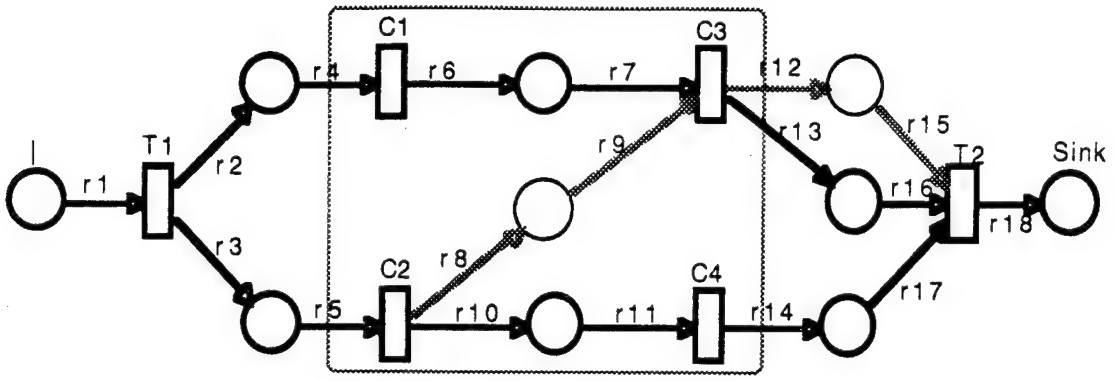


Figure 4.36 Fixed Structure FS2 for (a1, b2)

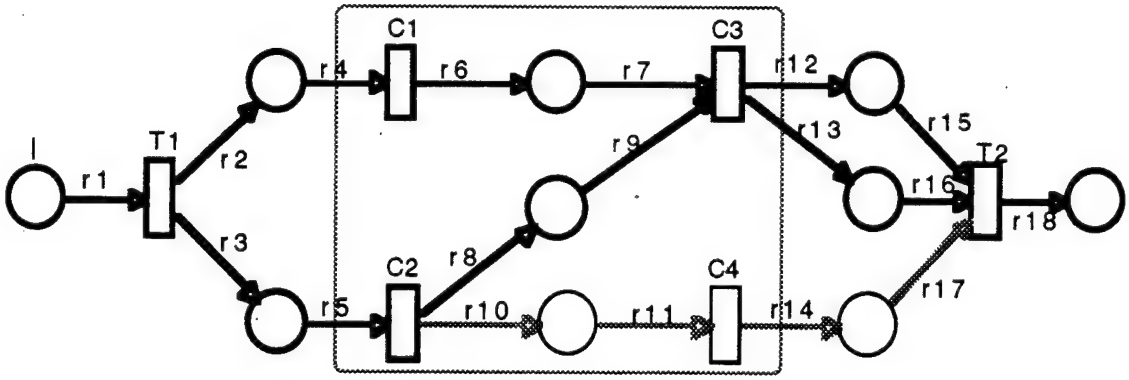


Figure 4.37 Fixed Structure FS3 for (a2, b1)

A variable structure VS is obtained by folding the four fixed structures. VS is denoted as VS = {FS1 under Input Situation (a1, b1), FS2 under Input Situation (a1, b2), FS3 under Input Situation (a2, b1), FS1 under Input Situation (a2, b2)}. Thus the variable structure varies for different inputs.

Mathematical Representation of the System Layer

An arc is represented by a function defined on the Feasible Input Domain **fi:** ***I*** --> {0, 1}. **fi**(v) = 0 means the token with color v cannot traverse this arc; **fi**(v) = 1 means the token with color v can traverse the arc. The function defined on the arc can be represented by an array if the elements in ***I*** are finite and discrete.

A transition/component is also represented by a function defined on the Feasible Input Space **ft:** ***I*** --> **FS**, where **FS** is the set of alternative tasks the transition/component may choose from. The

firing modes of a transition can be represented by an array, if the elements in **FS** are finite and discrete.

If all the arrays of the arcs and transitions/components have been defined, a variable structure is defined. Therefore, we can represent the variable structure by an incidence matrix, except that the entries of the matrix are arrays instead of scalars.

The variable structure represented by arrays can be unfolded into a set of fixed structures with each fixed structure corresponding to one Input Situation, and vice versa. If we fold the three distinct fixed structures described in Figures 4.35 through 4.37, we can obtain a variable structure with its arcs and transitions annotated by arrays. For example, arc r11 and transition C3 are annotated as shown in Figure 4.38. It is obvious that arc r11 is included in the fixed structure corresponding to Input Situation (a1, b2), but not included in fixed structures corresponding to other Input Situations. Transition C3 executes function f3 under all Input Situations.

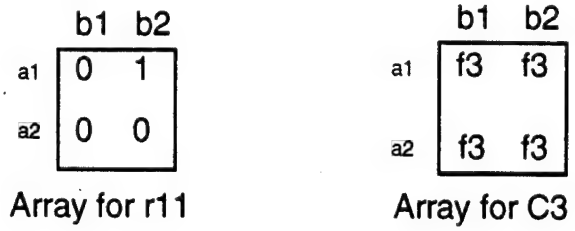


Figure 4.38 Arrays for Arc r11 and Transition C3

4.3.3 Coordination Constraint

Definition 4.8: A *Partition Process* ***P*** on a set **S** is a process of creating a set $\boldsymbol{P} = \{S_1, S_2, ..., S_n\}$ where S_1, S_2, ..., and S_n are subsets of **S**. For all elements in ***P***, $S_i \cap S_j = \phi$, if $i \neq j$, where $S_i \in \boldsymbol{P}$ and $S_j \in \boldsymbol{P}$. $S_1 \cup S_2 \cup ... \cup S_n = \mathbf{S}$. $S_i \in \boldsymbol{P}$, i = [1, n], is called *a partition* of ***S***.

Definition 4.9: Let two Partition Processes be defined on set **S**, $\boldsymbol{P}=\{S_{p1}, S_{p2} ..., S_{pn}\}$ where S_{p1}, S_{p2}, ..., S_{pn} are disjoint subsets of **S**, and $\boldsymbol{Q}=\{S_{q1}, S_{q2}, ..., S_{qm}\}$ where S_{q1}, S_{q2}, ..., S_{qm} are disjoint subsets of **S**. The operation *MERGE* is defined as $\boldsymbol{R} = \boldsymbol{P}$ MERGE $\boldsymbol{Q}$ if $\boldsymbol{R} = \{S_{r1}, S_{r2}, ..., S_{rl}\}$ is a set of subsets of **S** generated by the process defined in Figure 4.39.

Proposition 4.9: If $\boldsymbol{R} = \boldsymbol{P}$ MERGE $\boldsymbol{Q}$, then ***R*** is a Partition Process.

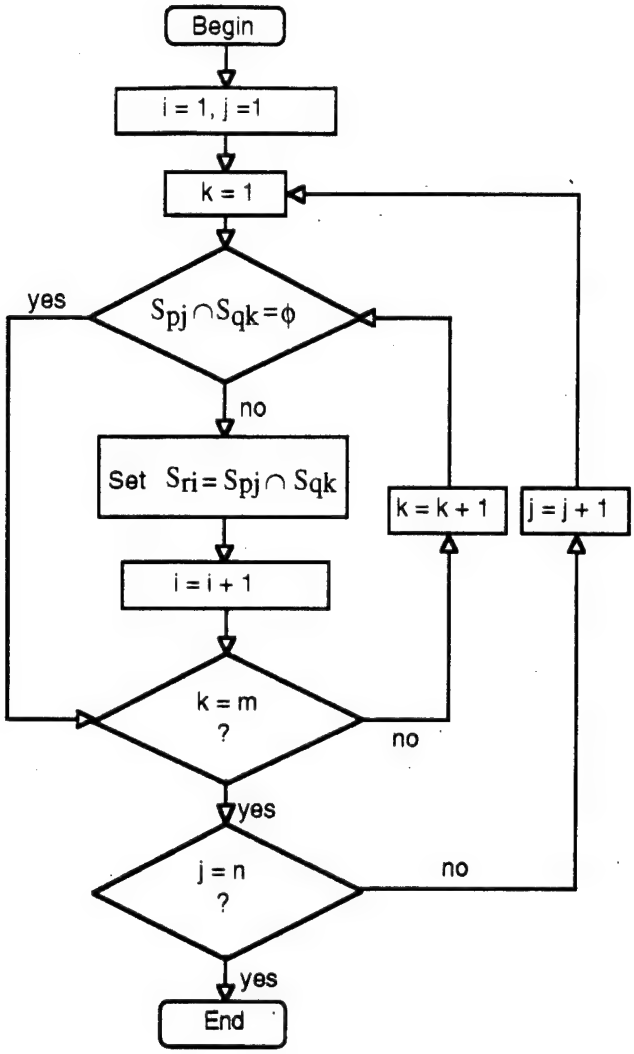


Figure 4.39 Define ***R*** = ***P*** MERGE ***Q***

Proof:

Let $S_{ri} \cap S_{rj} = (S_{pi1} \cap S_{qj1}) \cap (S_{pi2} \cap S_{qj2})$. If $i \neq j$, then either $i1 \neq i2$ or $j1 \neq j2$ (If $i1 = i2$ and $j1 = j2$ both stand, then $S_{ri} = S_{rj}$ means that $i = j$, contradictory with the assumption $i \neq j$). Suppose $i1 \neq i2$. $S_{ri} \cap S_{rj} = (S_{pi1} \cap S_{pi2}) \cap (S_{qj1} \cap S_{qj2}) = \phi$.

$S_{r1} \cup S_{r2} \cup \ldots \cup S_{rl} = \cup_{j=1,n}[\cup_{k=1,m}(S_{pj} \cap S_{qk})] = \cup_{j=1,n}[S_{pj} \cap (\cup_{k=1,m} S_{qk})] = \cup_{j=1,n}[S_{pj} \cap S] = \cup_{j=1,n} S_{pj} = S$

For example two Partition Processes are defined as $\boldsymbol{P}=\{S_{p1}, S_{p2}, S_{p3}\}$ (with $S_{p1}=\{a1, a2\}$, $S_{p2} = \{a3, a4, a5, a6\}$, $S_{p3} = \{a7\}$) and $\boldsymbol{Q}=\{S_{q1}, S_{q2}, S_{q3}\}$ (with $S_{q1} = \{a1, a2\}$, $S_{q2} =$

{a3, a4, a5}, S_{q3} = {a6, a7}). When the procedure defined in Figure 4.39 is followed, it forms a set {S_{r1}, S_{r2}, S_{r3}, S_{r4}} (with S_{r1} = {a1, a2}, S_{r2} = {a3, a4, a5}, S_{r3} = {a6} and S_{r4} = {a7}).

Let us define a function ***F*** on set **S** partitioned by a Partition Process into ***P*** = {S_1, S_2, ...,S_n}, and let ***F*** have the following properties: ***F***(x) = ***F***(y) if and only if x∈ S_i, y∈ S_j and i = j. That is, the function ***F*** evaluates to the same value for all the elements in the same partition, and evaluates to different values for elements in different partitions. Actually, function ***F*** is nothing more than another representation of a Partition Process. Given the function ***F*** , we can recreate the Partition Process by simply dividing set S according to the value of x (x∈**S**).

Proposition 4.10: A function ***F*** for a Partition Process ***P*** is defined on set **S**. A system (system R) is defined by a function **R**(z), which means that if the input to system R is z, then the output of the system is **R**(z). Let x, y ∈ S, and let ***F***(x) and ***F***(y) be inputs to system R, then the output of system R are **R**[***F***(x)] and **R**[***F***(y)], respectively. If x, y ∈ S_i, then **R**[***F***(x)] = **R**[***F***(y)].

Proof:

x, y ∈ S_i ==> ***F***(x) = ***F***(y). ==> **R**[***F***(x)] = **R**[***F***(y)]

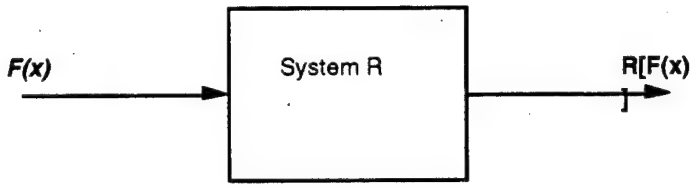


Figure 4.40 A System Described by **R**[z]

Proposition 4.11: Let n functions be defined on set **S:** ***F1*** on Partition Process ***P1; F2*** on Partition Process ***P2; ...; Fn*** on Partition Process ***Pn.*** A system (system R) is defined by a function **R**(z1, z2, ..., zn), which means that if z1, z2, ..., zn are inputs, then **R**(z1, z2, ..., zn) is the output of the system.

Let x, y ∈ **S**, and let ***F1***(x), ***F2***(x), ..., ***Fn***(x) and ***F1***(y),***F2***(y),...,***Fn***(y) be inputs of a system, and **R**[***F1***(x), ***F2***(x), ..., ***Fn***(x)] and **R**[***F1***(y),***F2***(y),...,***Fn***(y)] be the outputs of the system, respectively (Figure 4.41).

Let $P = P1$ MERGE $P2$ MERGE $P3$ ....MERGE Pn. Denote Partition Process $P = \{S_1, ..., S_n\}$.

If x, y $\in S_i$, then $\mathbf{R}[F1(x), F2(x) ..., Fn(x)] = \mathbf{R}[F1(y), F2(y), ..., Fn(y)]$.

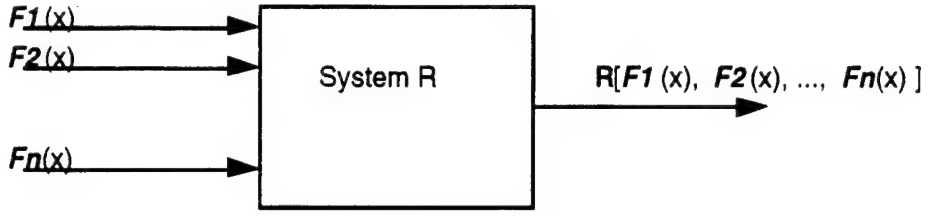


Figure 4.41 A System Described by R[z1, z2, ...,zn]

Proof:

x, y $\in S_i$ ==> $S_i = S_{1p_1} \cap S_{2p_2} \cap ... \cap S_{np_{1n}}$ (Where $S_{1p_1} \in P1$, $S_{2p_2} \in P2$, ..., $S_{np_n} \in Pn$). ==> x, y $\in S_{1p_1}$; x, y $\in S_{2p_2}$; ... ; x, y $\in S_{np_n}$. ==> $F1(x) = F1(y)$; $F2(x) = F2(y)$; ...;$Fn(x) = Fn(y)$. ==> $\mathbf{R}[F1(x), F2(x), ..., Fn(x)] = \mathbf{R}[F1(y), F2(y), ..., Fn(y)]$.

In the organization, each transition (or component) is connected to some sensors through some paths. Therefore, it is possible that a component only accesses a limited set of alphabets. This situation can be modeled by a Partition Process P defined on the Feasible Input Domain for a component. Let the input to the component be $F(x)$, $x \in \mathbf{S}$; the component is not able to make different responses if x is located within the same partition of P. For example, let an aircraft have speed v and size s as its characteristics, denoted as x = (v, s), where $v \in V = \{5, 6, 7\}$ and $s \in S = \{$small, medium, large$\}$. If a speed sensor can only sense the speed of the aircraft, whatever its size, we may partition the space V*S into three subsets: {(5, small), (5, medium), (5, large)}, {(6, small), (6, medium), (6, large)}, and {(7, small), (7, medium), (7, large)}. Function $F(x)$ can be considered as the output of the speed sensor. If $F(x)$ is the input of a component in the organization, the component can not respond differently to $F(x1)$ and $F(x2)$ if $F(x1) = F(x2)$ (or x1 and x2 are in the same partition) according to the proposition.

If a component in an organization has multiple inputs, all information from the inputs can be used to distinguish the current Input Situation according to the proposition.

In the variable structure design problem, for each Input Situation, we usually can get a set of fixed structures which is able to perform the required tasks. Take one structure from each set, and fold them together. A variable structure is obtained. For example, suppose the Feasible Input Domain is A * B where A = {a1, a2} and B = {b1, b2}. As shown in Figure 4.42, for Input Situation (a1,b1), there is a set of fixed structure {FS11, FS12, FS13, FS14}, for Input Situation (a1,b2), a set of fixed structure {FS21, FS22, FS23}, for Input Situation (a2, b1), a set of fixed structure {FS31, FS32, FS33, FS34, FS35}, and for Input Situation (a2, b2), a set of fixed structure {FS41, FS42, FS43, FS44}. Take FS12 for Input Situation (a1,b1), take FS21 for Input Situation (a1,b2), take FS34 for Input Situation (a2,b1), take FS43 for Input Situation (a2, b2). A variable structure VS is obtained by folding these four fixed structures.

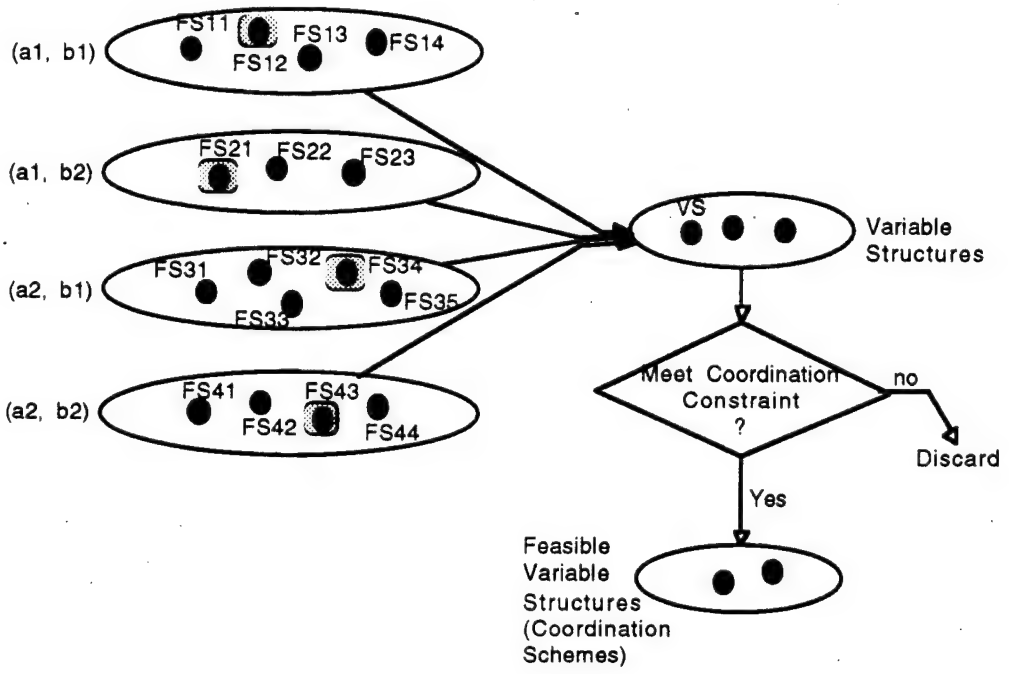


Figure 4.42 Find Feasible Variable Structures

Because the selection of a fixed structure from each set to form a variable structure is arbitrary, if there are m Input Situations and there is a set of n_i fixed structures corresponding to the ith Input Situation, then there will be $\prod_{i=1}^{m} n_i$ variable structures. But not all these variable structures are realizable. Only the variable structures which meet the Constraint are feasible.

Coordination Constraint

Let each component of a system be associated with a Partition Process ***P*** on the Feasible Input Space. ***P*** is defined according to the accessed information by that component, such that the tokens (whose colors are represented by an Input Situations) are indistinguishable if they are located in the same partition.

A component is said to meet the Coordination Constraint if it does not respond differently to the tokens whose colors (represented by an Input Situation) are located in the same partition.

Variable interactions between two components C1 and C2 must be coordinated on sources of information that are accessed jointly by them (directly or indirectly). The stage C1 must determine, based on some information it has processed, whether it has to send a message to C2. Similarly, the role that contains C2 must infer from some of the information it has already received, whether or not it must wait for a message or information from C1 before initiating process C2.(Demaël, 1989). The above requirements are termed as Coordination Constraint. Generalizing these requirements, a transition or component in the CPN representation is said to meet the coordination constraint if it is not required to respond differently for situations which are indistinguishable to it. A variable structure is said to meet the coordination constraint if all its components meet the coordination constraint. A structure satisfying the constraint is feasible in the sense that it can be realized.

The Algorithm

Figure 4.42 describes the procedure to find feasible variable structures. At the first stage, variable structures are constructed by arbitrarily choosing a fixed structure from each set of an Input Situation. At the second stage, an algorithm is used to check the Coordination Constraint for each generated variable structure. Only the variable structures which meet the Coordination Constraint are retained and stored as feasible variable structures. The Coordination Constraint is like a filter which rules out the unrealizable variable structures. In this section, an algorithm to check the Coordination Constraints is proposed.

The Coordination Constraint described in the previous section indicates that the entries of arrays of output arcs and the component must be the same within a partition of the Partition Process of a component. The partition processes for the input arcs of a component with multiple inputs are dealt with a different scheme. For components with multiple inputs, the entries of every array of the input arcs must be the same within a partition of all the Partition Process associated with the input

arcs. The process is repeated for every input arc of a component with multiple inputs. Figure 4.43 shows examples of checking the Coordination Constraint. The arrays in Figure 4.43 could be of an arc or component. The solid lines partition each array into six partitions. The entries in each partition of the first array are the same; consequently, the Coordination Constraint is satisfied. In the second case, one of the partitions has two different elements. There the Coordination Constraint is not satisfied. Therefore, if we can find the Partition Processes for all the components in the organization, and find that they meet the Coordination Constraint, a realizable or feasible variable structure is obtained.

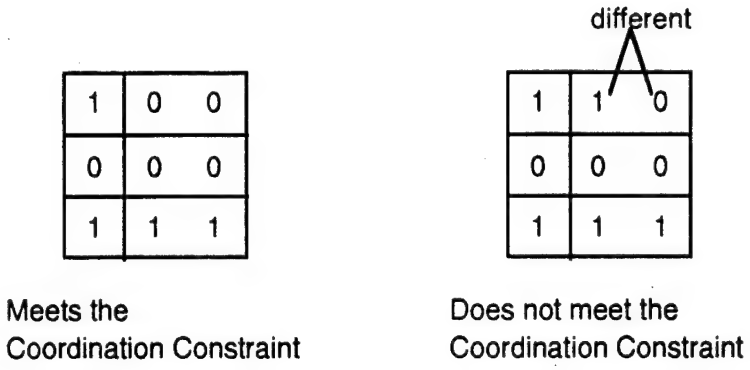


Figure 4.43 The Coordination Constraint

Before introducing the algorithm, the concept of a Partition Process for an arc, and three basic procedures for deriving the Partition Process need to be introduced. The Partition Process of an arc is the information which an arc carries. If the component has m input arcs with their Partition Processes denoted as ***P1, P2, ...Pm***, and the Partition Process of the component is denoted as ***Q***, then ***Q*** = ***P1*** MERGE ***P2*** MERGE ... MERGE ***Pm***. Figure 4.44 shows an example of deriving the Partition Process for a component given the component array and the input arc annotations.

Given the Partition Processes for a component and the array for an output arc, the Partition Process for an output arc from a component is produced by erasing the part of the partition that is within a zero region in the array of the output arc. In other words, the partitions whose entries are 0 are put together into one partition. Figure 4.45 shows an example. It should be emphasized that the two zero regions are in one partition of the arc. The information which a component accesses is carried by the tokens coming to this component. Zero regions in the array of an arc mean that no token can go through this arc. Therefore, the information contained in the token cannot pass through either.

This is the reason why the partition in zero regions need to be removed. For multiple output arcs of a component, the Partition Process for each output arc is generated independently.

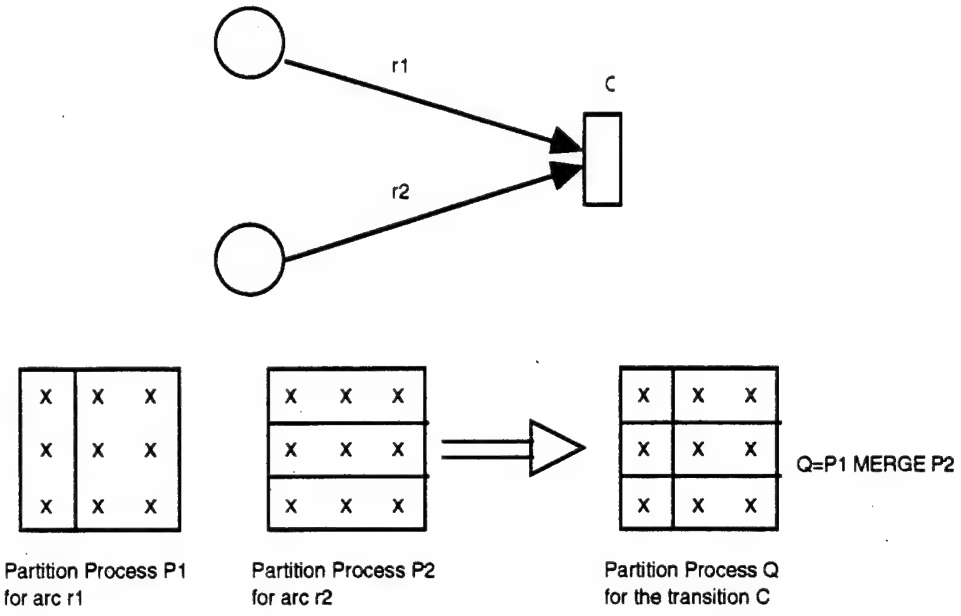


Figure 4.44 Derivation of the Partition Process for a Component

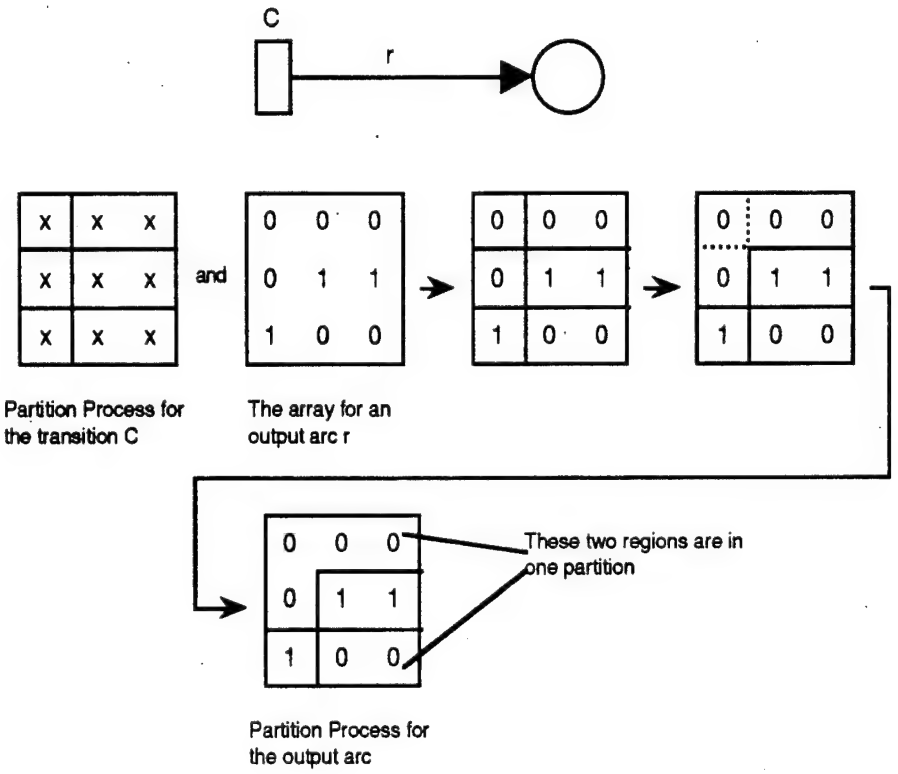


Figure 4.45 The Partition for an Output Arc

Since the nets analyzed in this thesis are marked graphs, if both sink and source are merged and under compact form, a place in an organization has only one input arc and one output arc. Generating the Partition Process of the output arc of a place given the Partitions Process of the input arc of the place is essentially an identity operation.

The flow chart of the proposed algorithm is shown in Figure 4.46. An example is provided to illustrate this algorithm in the reaming part of this section. Figure 4.47 shows the System Layer of an organization. Arrays for arcs and components are defined in Figure 4.48. Suppose the Feasible Input Domain is A * B, where a = {a1, a2, a3} and B = {b1, b2, b3}. Components C1 and C2 are sensors. Suppose C1 can only sense alphabet letters in set A and C2 can only sense alphabet letters in set B.

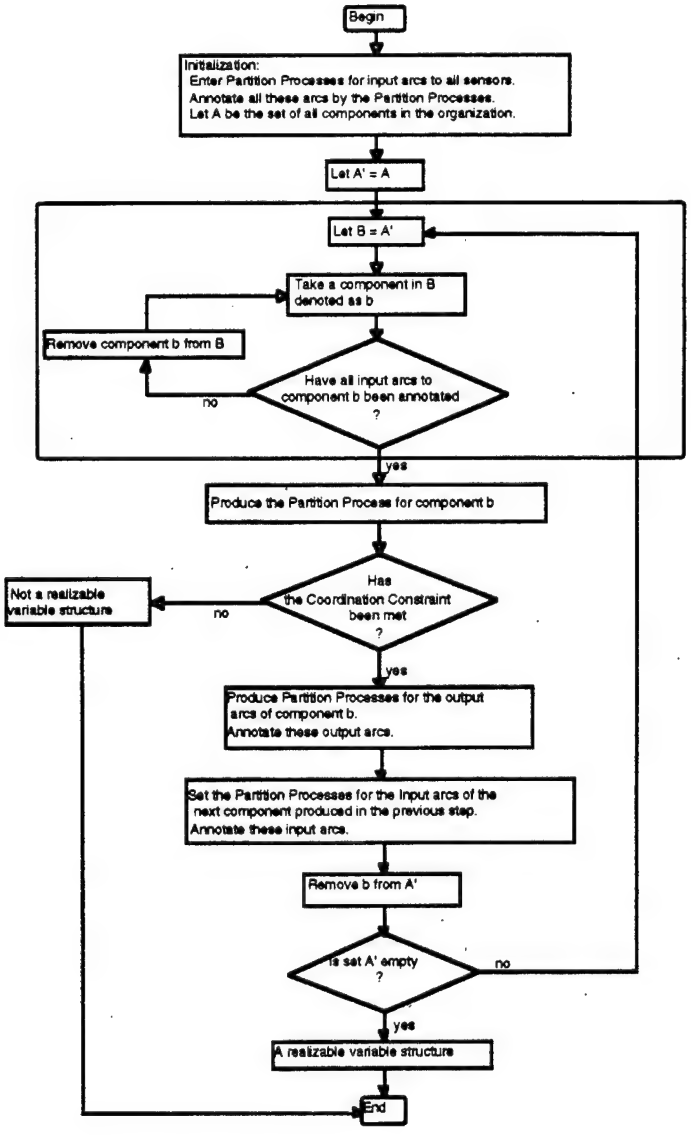


Figure 4.46 An Algorithm for Checking the Coordination Constraint

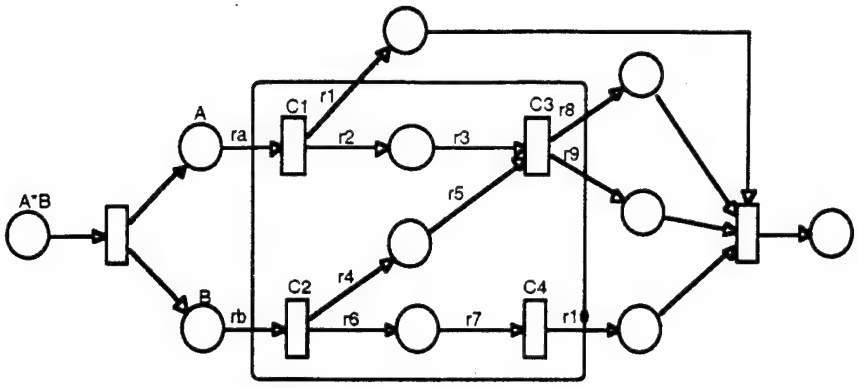


Figure 4.47 The System Layer of an Organization

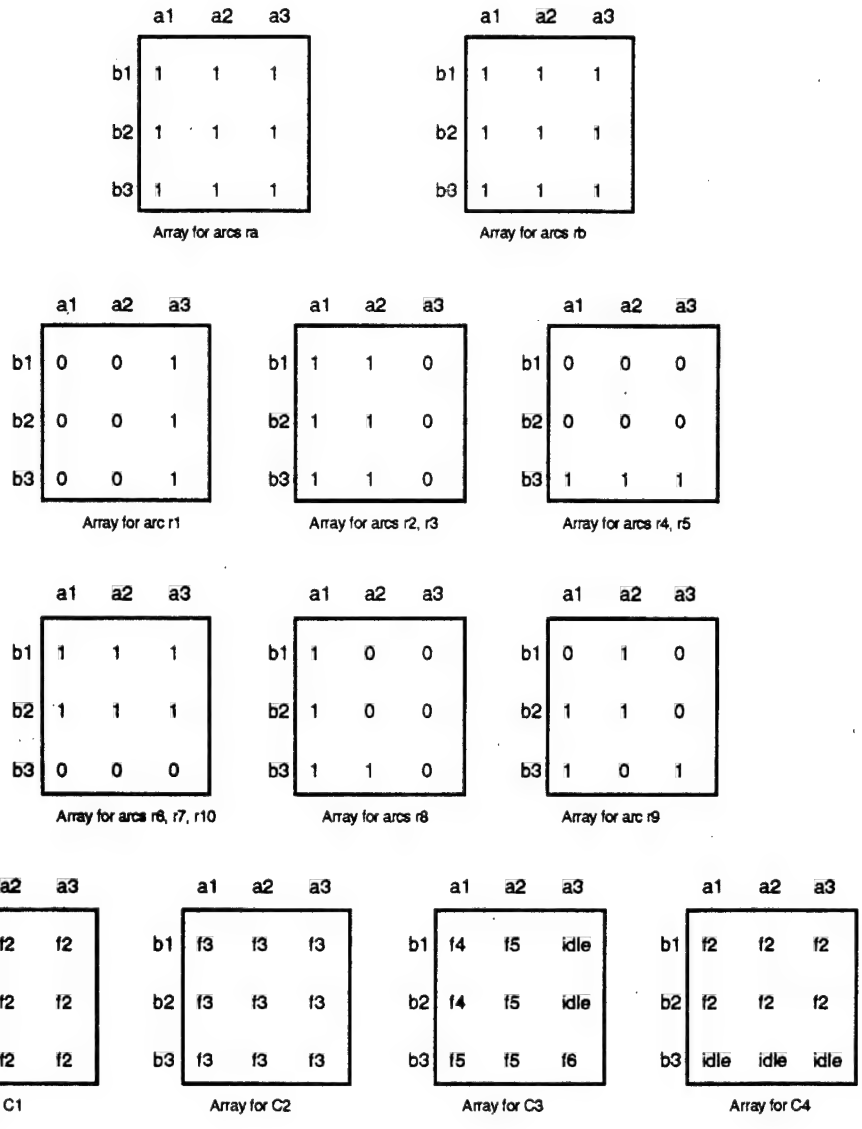


Figure 4.48 Arrays for the Arcs and Components

The first step in the algorithm is the initialization part. The Partition Process for each input arc of each sensor needs to be entered by the user as initial conditions. By analyzing the nature of the sensors C1 and C2, we determine that the Partition Processes for the input arcs to the sensors are as shown in Figure 4.49. The initial part of the algorithm annotates arcs ra and rb, and defines set A = {C1, C2, C3, C4}. A' is also set to be A' = {C1, C2, C3, C4}.

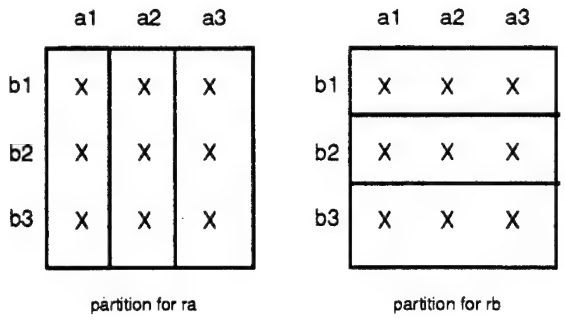


Figure 4.49 Partition Processes for Arcs ra and rb

The part of the algorithm outlined in Figure 4.47 is used to identify components that have not been traversed, but whose input arcs have all been annotated. Suppose C1 is selected first (ra is annotated) and go to the next steps.

Given the Partition Process for arc ra, the Partition Process for C1 is derived, and the Coordination Constraint is checked (All the arrays of the output arcs and the component are checked). The result are illustrated in Figure 4.50. It is found that C2 meets the Coordination Constraint. The algorithm continues.

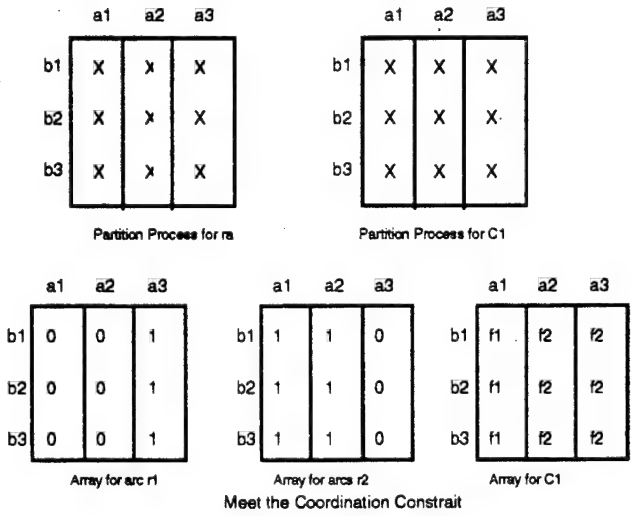


Figure 4.50 Partition Process for C1 and Checking the Coordination Constraint

Given the Partition Process for C1, the Partition Processes have been derived for output arcs r1 and r2. The result is shown in Figure 4.51. The Partition Process for r2 is passed to the input arc r3 for component C3 (It is an identity operation) and r3 has the same Partition Process as r2. During the procedure, arcs r1, r2 and r3 are annotated.

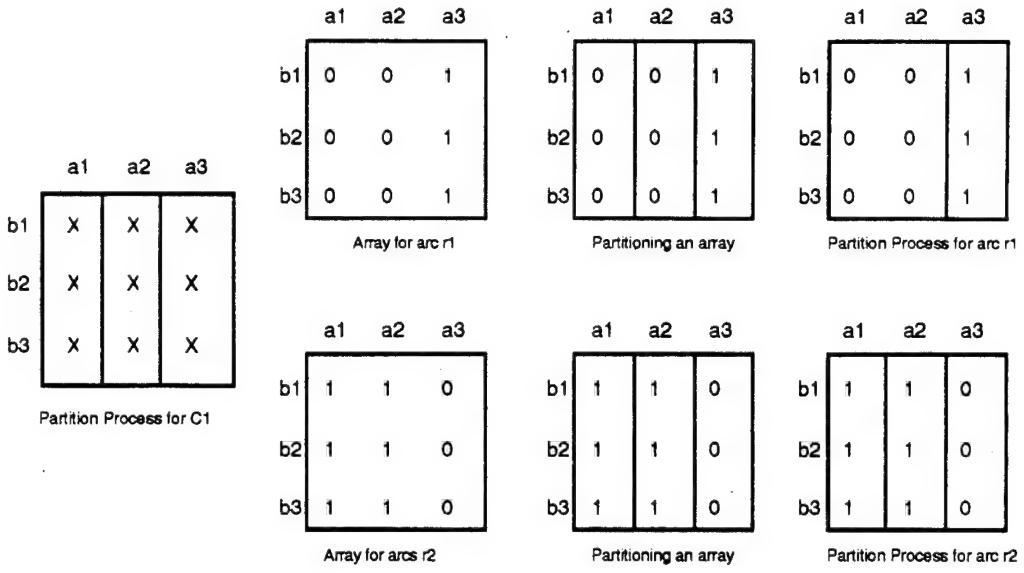


Figure 4.51 Finding the Partition Processes for Output Arcs of C1

C1 is removed from A', and A' is set to {C2, C3, C4}. Since A' is not empty, go back to the loop.

Next, component C2 is selected (because rb is annotated). The Partition Processes for C2 is derived and is found to meet the Coordination Constraint. The Partition Processes for arcs r4, r6, r5, and r7 are derived and annotated. These results are shown in Figure 4.52. C2 is removed from A' (A' = {c3, c4}). The algorithm continues.

Since the component C3 has multiple inputs the partition process associated with arc r3 is applied to partition the array associated with r5 and vice versa. A simple inspection reveals the fact that several partitions in both arrays have differenct entries, making the variable structure infeasible. The same result can be achived by generating the partition Process for C3 and checking the partitioned array for inconsistencies. The Partition Process for component C3 can be generated by applying the MERGE operator to the Partition Processes for arcs r3 and r5 (shown in Figure 4.53). The Coordination Constraint is checked for C3. We find that one partition of the Partition Process of r9 has different entries. Therefore, it does not meet the Coordination Constraint (Figure 4.54). This is not a realizable variable structure.

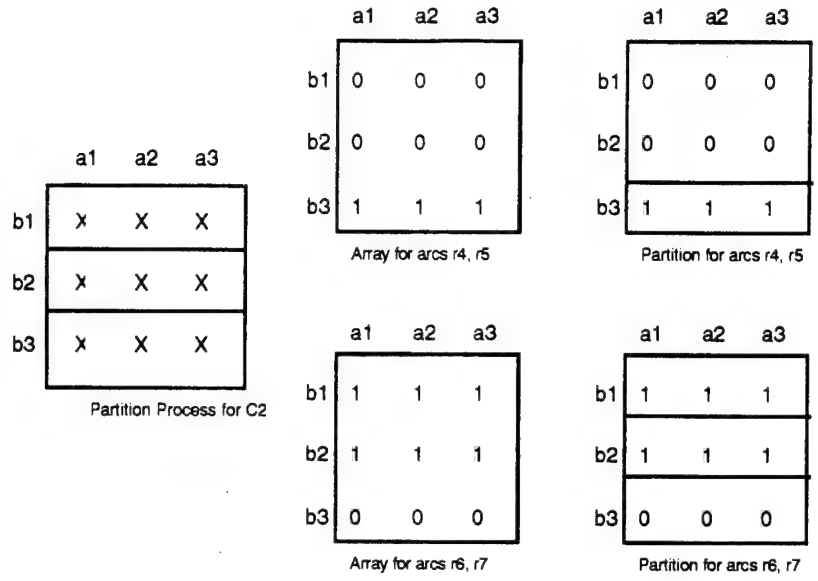


Figure 4.52 The Partition Processes for C2, r4, r5, r6 and r7

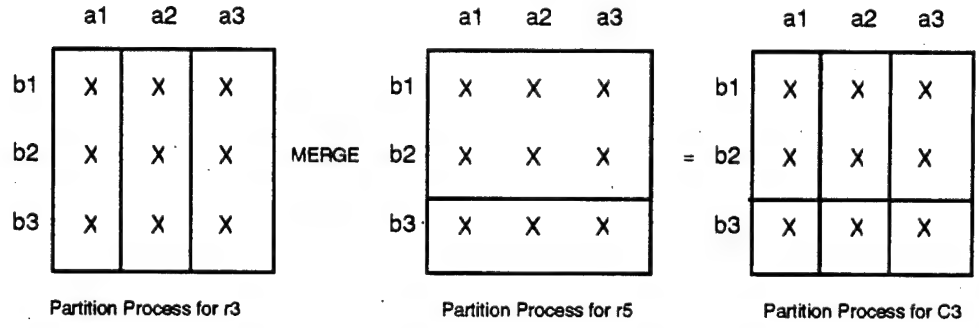


Figure 4.53 Generating the Partition Process for C3

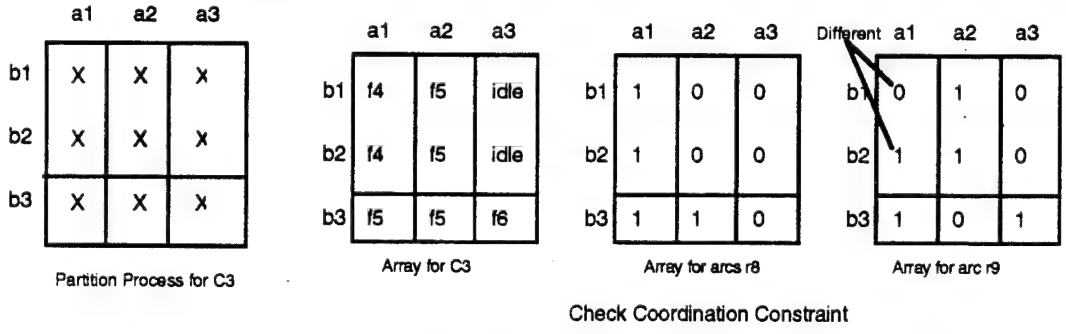


Figure 4.54 Checking Coordination Constraint

Let us take a close look at Figure 4.47 (the System Layer) and Figure 4.48 (the arrays for all arcs and components). Component C1 only senses the alphabet in color set A. C1 can make the decision to send token a3 into the upper place though arc r1 and send token a1 and a2 into the lower place through arc r2 (according to the arrays in Figure 4.48). Component C3 can receive tokens a1 or a2, but never a3. If C3 receives a token a1, the Input Situation could be (a1, b1), (a1, b2) or (a1, b3), but it cannot be sure exactly which one it is, based on this information. Component C2 sends token b3 to component C3 but sends token b1 or b2 to component C4. If C3 receives a token b3, it knows the current Input Situation is (a1, b3), (a2, b3) or (a3, b3), based on this information.

By combining these two pieces of information, C3 can deduce:

- If a token a1 comes from r3 and no token comes from r5, it can deduce that the current Input Situation might be (a1, b1) or (a1, b2), but it is not sure exactly which one.
- If a token a1 comes from r3 and a token b3 comes from r5, it can deduce that the current Input Situation is (a1, b3) for sure.
- If a token a2 comes from r3 and no token comes from r5, it can deduce that the current Input Situation might be (a2, b1) or (a2, b2), but it is not sure exactly which one.
- If is a token a2 comes from r3 and a token b3 comes from r5, it can deduce that the current Input Situation is (a2, b3) for sure.
- If is no token comes from r3 and a token b3 comes from r5, it can deduce that the current Input Situation is (a3, b3) for sure.
- If is no token comes from r3 and r5, it can deduce that the current Input Situation might be (a3, b1) or (a3, b2), but he is not sure exactly which one (This based on the assumption that C3 knows there is an input).

Take a look at array for arc r9: it requires component C3 to send a token for Input Situation (a1, b2) and not to send a token for Input Situation (a1, b1). But for both Input Situations, C3 receives a token a1 from r3 and no token from r5. It cannot distinguish whether it is Input Situation (a1, b2) or (a1, b1). Therefore the whole variable structure is not realizable.

4.3.4 Conclusions

Folding fixed structures into a realizable variable structure is an important issue in organization design applications. A component must have the necessary information to vary its structure or process an incoming information, i.e., if the inputs to the component are the same for two Input Situations, the component should not be able to produce different responses for these two Input

Situations. This property is expressed as the Coordination Constraint which is an extension of a constraint (Constraint R10) proposed by Demaël (1989). The constraint has been addressed in detail and an algorithm has been developed for checking it.

CHAPTER V

CONCLUSIONS AND MATTERS OF RECORD

5.1 CONCLUSIONS

This three year effort was, in a sense, a continuation and expansion of the research started under the "Distributed Tactical Decision Making" initiative by both co-principal investigators. A deliberate attempt was made to synthesize approaches from cognitive sciences, computer sciences, and system theory. The results of that effort have been presented in Chapters II, III, and IV. However, this is not the complete record of the research. Selected results were presented so that this final report is self-contained. A more complete record of the research is obtained, if one considers all the publications that have been produced as a result of this project. This list is presented in section 5.2.

During the last year of research, a set of new problems were identified whose solution is essential for pursuing the development of a theory of coordination in variable structure organizations. In the past, it was possible to adapt already existing results in the theory of discrete event systems, Petri Nets, and Colored Petri Nets, so that they can be used in the context of the analysis and design of decision making organizations. This is not the case any more. It has become necessary to extend the existing theory and develop new algorithms for its implementation. This has led to several basic research topics, appropriate for Master's and Ph.D. theses, that are currently under way. Since these theses are in progress, their description has not been included in this report. However, the expected results to be documented in the future will owe their existence to this project. The list of students who have participated and contributed to this project is given in section 5.3.

5.2 DOCUMENTATION

5.2.1 Theses

S. A. K. Zaidi, "On the generation of Multilevel, Distributed Intelligence Systems using Petri Nets," MS Thesis, Report GMU/C3I-113-TH, C3I Center, George Mason University, Fairfax, VA, November 1991. (Advisor: Prof. Levis)

Z. Lu, "Coordination in Distributed Intelligence Systems," MS Thesis, Report GMU/C3I-120-TH, C3I Center, George Mason University, Fairfax, VA, May 1992. (Advisor: Prof. Levis)

A. Sadigh - MS Thesis due February 1994. (Advisor: Prof. Lehner)

Z. Jin - MS Thesis due February 1994. (Advisor: Prof. Levis)

A. Zaidi - PhD Thesis due December 1994. (Advisor: Prof. Levis)

T. Zhang - PhD Thesis due June 1995. (Advisor: Prof. Levis)

5.2.2 Papers in Refereed Journals

Perdu, D. M., and A. H. Levis, "Requirements Determination Using the Cube Tool Methodology and Petri Nets," *IEEE Trans. om Systems Man and Cybernetics,* **SMC-23**, No. 5, Sept./Oct. 1993

Lehner, P.E., and Ulvila, J. W., "A Note on the Application of Classical Statistics to Evaluating the Knowledge Base of an Expert System," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 23, No. 2, 1993, pp. 563-569.

Levis, A. H., Moray, N. and Baosheng Hu, "Task Allocation and Discrete Event Systems," *Automatica,* February 1994.

Johannsen, G., A. H. Levis, and H. Stassen, "Theoretical Models in Man - Machine Systems and their Experimental Validation," *Automatica,* February 1994.

Demaël, J. J. and A. H. Levis, "On Generating Variable Structure Architectures for Decision Making Systems," *Information and Decision Technologies,* 1994.

5.2.3 Chapters in Books

Levis, A. H., "Modeling and Design of Distributed Intelligence Systems," in *Introduction to Autonomous and Intelligent Control,* P. J. Antsaklis and K. M. Passino, Eds., Kluwer Publishers, Boston, MA, 1993.

Levis, A. H., "A Colored Petri Net Model of Command and Control Nodes," in *Toward a Science of Command Control and Communications,* Carl R. Jones, Ed., AIAA Press, Washington, DC,1993.

Monguillet, J. M., and A. H. Levis, "Modeling and Evaluation of Variable Structure Organizations," in *Toward a Science of Command Control and Communications,* Carl R. Jones, Ed., AIAA Press, Washington, DC, 1993.

Jin, V. Y., and A. H. Levis, "Impact of Organizational Structure on Team Performance: Experimental Findings," in *Toward a Science of Command Control and Communications,* Carl R. Jones, Ed., AIAA Press, Washington, DC, 1993.

Levis, A. H. "Human Interaction with Decision Aids: A Mathematical Approach," in *Human/ Technology Interaction in Complex Systems,* Vol. 7, W. B. Rouse, Ed., JAI Press; in press; to appear in 1994.

Zaidi, S. A. K., and A. H. Levis, "Algorithmic Design of Distributed Intelligence Systems," in *Intelligent Control Systems: Theory and Practice,* M. M. Gupta and N. K. Sinha, Eds., IEEE Press, in press, to appear in 1994.

5.2.4 Papers or Reports in Non-refereed Journals

Lam, N.T. and Lehner, P.E. "A Quantitative Approach to Predicting the Usefulness of a Decision Aid," *Proc. 1992 IEEE International Conference on Systems, Man, and Cybernetics,* October 1992.

Lu, Zhuo and A. H. Levis, "Coordination in Distributed Decision Making," *Proc. 1992 IEEE International Conference on Systems, Man, and Cybernetics,* October 1992.

Zaidi, S. A. K. and A. H. Levis, "Algorithmic Design of Multilevel Organizational Structures," *Proc. 1992 IEEE International Conference on Systems, Man, and Cybernetics,* October 1992.

Hoh, Y. S., D. M. Perdu, and A. H. Levis, "A Methodology of Evaluating the Copernicus Architecture Using Colored Petri Nets," *Proc. 1993 Symposium on C2 Research,* National Defense University, Ft. McNair, Washington, DC, June 1993.

Ray, B. and A. H. Levis, "Functional Architecture for Crisis Management at the National Military Command Center," *Proc. 1993 Symposium on C2 Research,* National Defense University, Ft. McNair, Washington, DC, June 1993 .

Perdu, D. M., S. A. K. Zaidi, A. Sadigh, P. Lehner, and A. H. Levis, "On a Methodology for Team Design Using Influence Diagrams," *Proc. 1993 Symposium on C2 Research,* National Defense University, Ft. McNair, Washington, DC, June 1993.

Sadigh, A., and Lehner, P. E., "The Complexity of Near-Optimal Decision Procedures," *Proc. 1993 Symposium on C2 Research,* National Defense University, Ft. McNair, Washington, DC, June 1993.

Lehner, P. E. and A. Sadigh, "Two Procedures for Compiling Influence Diagrams, " *Proc. 1993 Conf. on Uncertainty in AI,* Morgan Kaufmann, 1993.

5.3 RESEARCH PERSONNEL

The following persons participated in this effort during the three year period.

Prof Paul Lehner	GMU - Co-Principal Investigator
Prof. Alexander H. Levis,	GMU - Co-Principal Investigator
Mr. Didier Perdu	GMU - Graduate Student (PhD)
Mr. Lee Wagenhals	GMU - Graduate Student (PhD)
Mr. Syed Abbas K. Zaidi	GMU - Graduate Research Assistant (PhD)
Ms. Tong Zhang	GMU - Graduate Research Assistant (PhD)
Ms. Zhenyi Jin	GMU - Graduate Research Assistant (MS)
Ms. Azar Sadigh	GMU - Graduate Research Assistant (MS)

Mr. Mathew Christian	GMU - Graduate Research Assistant (MS)
Mr. N. Thomas Lam	GMU - Graduate Student (PhD)
Mr. Zhuo Lu	GMU - Graduate Research Assistant (MS received)
Mr. Bhashyam Nallappa	GMU - Graduate Research Assistant (MS)
Mr. Diwakar Prabhakar	GMU - Graduate Research Assistant (MS received)
Mr. Ali R. Shah	GMU - Graduate Research Assistant (MS received)
Mr. Mir-Masood Seyed-Solorforough	GMU - Graduate Research Assistant (PhD)

During the early stages of this research, a team from Decision Sciences Consortium (DSC) worked on the project under subcontract to GMU. However, the company was sold and the team was dispersed. Their contribution, however, is gratefully acknowledged.

Dr. Kent Hull	DSC
Dr. Martin Tolcott	DSC - Consultant
Dr. Theresa Mullin	DSC
Dr. Michael O'Connor	DSC - P.I. of subcontract
Mr. William Roman	DSC - Programmer
Mr. Steve Saks	DSC - Programmer
Dr. Michael Donnell	Consultant

REFERENCES

American National Standard Institute, 1970. Flowchart Symbols and their Usage in Information Processing. X3.5-1970.

Andreadakis, S.K., and Levis, A.H., 1988. Synthesis of Distributed Command and Control for the Outer Air Battle. *Proc. 1988 Symposium on Command and Control Research.* Science Applications International Corp., McLean, Virginia.

Andreadakis, S.K., 1988. Analysis and synthesis of decision-making organizations. Ph.D. Thesis, Report LIDS-TH-1740, Laboratory for Information and Decision Systems, M.I.T., Cambridge, MA.

Balbes, R., and Dwinger, P., 1974. *Distributive Lattices.* University of Missouri Press, Columbia, Missouri

Birkhoff, G., 1984. *Lattice Theory.* American Mathematical Society, Providence, RI.

Boettcher, K.L., and Levis, A.H. 1982. Modeling the interacting decision maker with bounded rationality. *IEEE Trans. Syst., Man Cybern.*, **SMC-12**, pp. 334-344.

Brams, G.W., 1983. *Réseaux de Petri: Théorie et Pratique Tomes 1 et 2.* Masson, Paris

Buede, D., 1992. Superior design features of decision analytic software. *Computers and Operation Research*, **19**(1), pp. 43-58.

Carré, B., 1979. *Graphs and Networks.* Oxford University Press, New York.

Conant, R.C., 1976. Laws of information which govern systems. *IEEE Trans. Syst., Man Cybern.*, **SMC-6**, 240-255.

Cooper, G.F., 1990. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, **42**, pp. 393-405.

Dean, T.L., and Wellman, M.P., 1991. *Planning and Control.* Morgan Kaufmann, San Mateo, California.

Demaël, J.J. 1989. On the generation of variable structure distributed architectures. MS thesis, Report LIDS-TH-1869, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Demaël, J.J., and Levis, A.H., 1990. On the generation of a variable structure airport surface traffic control system." *Proc. IFAC World Congress,* Tallinn, Estonia, USSR.

Edwards, W., 1968. Conservatism in human information processing. In Kleinmuntz, B., Ed., *Formal Representation of Human Judgment.* Wiley, New York.

Einhorn, H., and Hogarth, R., 1986. Judging probable cause. *Psychological Bulletin.* **99** pp. 3-19.

Fischhoff, B., 1975. Hindsight ≠ foresight: The effect of outcome knowledge on judgment under uncertainty. *Journal of Experimental Psychology:* Human perception and Performance 4, pp. 330-344.

Genrich, H., and Lautenbach, K., 1981. Systems modeling with high-level Petri Nets." *Theoretical Computer Science*, No.13, pp.109-136.

Grätzer, G. 1971. *Lattice Theory, First Concepts and Distributed Lattices.* W. H. Freeman and Company, San Francisco.

Hall, S.A., 1982. Information Theoretic Models of Storage and Memory. MS Thesis, Report LIDS-TH-1232, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Heckerman, D., Breese, J., and Horvitz, E., 1989. The compilation of decision models. *Proceedings of the 1989 Workshop on Uncertainty in Artificial Intelligence*, pp. 162-173.

Henrion, M., 1988. Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer, J.F., and Kanal, L.N., Eds.*Uncertainty in Artificial Intelligence 2.* North Holland, Amsterdam.

Hillion, H.P., 1986. Performance evaluation of decisionmaking organizations using timed Petri Nets. MS Thesis, Report LIDS-TH-1590, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Jensen, K., 1992. *Coloured Petri Nets.* Springer-Verlag, Berlin, Germany.

Jin, V.Y., 1990. On the effect of Organization Structure on Team Performance. Ph.D. Thesis, Report LIDS-TH-1976, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Kahneman, D., and Tversky, A., 1973. On the psychology of prediction. *Psychological Review,* **80** pp. 237-251.

Kahneman, D., Slovic, P., and Tversky, A., 1982. *Judgment under uncertainty: Heuristics and biases..* Cambridge University Press.

Lehner, P.E., and Adelman, L., 1990. Behavioral decision theory and its implications for knowledge engineering. *Knowledge Engineering Review.*

Lehner, P.E., and Sadigh, A., 1992. A procedure for compiling influence diagrams, *Proceedings of the 1992 Symposium on C2 Research,* Science Applications International Corp., McLean, VA.

Lehner, P.E., and Sadigh, A., 1993. Two procedures for compiling influence diagrams. *Proceedings of the 1993 Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, San Mateo, California.

Lehner, P.E., and Zirk, D.A., 1987. Cognitive factors in user/expert system interaction. *Human Factors*, **29**(1), pp. 97-109.

Levis, A.H. 1984. Information processing and decision making organizations: A mathematical description. *Large Scale Systems*, **7**, pp. 151-163.

Levis, A.H., 1988. Human organizations as distributed intelligence systems. *Proc. IFAC Symposium on Distributed Intelligence Systems.* Pergamon Press, Oxford, England.

Levis, A.H., 1992. Modeling and design of distributed intelligence systems" in *Introduction to Autonomous and Intelligent Control,* P. J. Antsaklis and K. M. Passino, Eds., Kluwer Publishers, Boston, MA.

Levis, A.H., and Boettcher, K.L., 1983. Modeling and analysis of teams of interacting decision makers with bounded rationality. *Automatica*, **9**(6), pp. 703-709.

Levis, A.H., 1992. A Colored Petri Net model of intelligent nodes. In *Robotics and Flexible Manufacturing Systems,* Gentina, J.C., and Tzafestas, S.G., Eds., Elsevier Science Publishers B.V., North-Holland.

Louvet, A.-C., Casey, J.T., and Levis, A.H., 1988. Experimental investigation of the bounded rationality constraint. In*Science of Command and Control: Coping with Uncertainty.* Johnson, S.E., and Levis, A. H., Eds. AFCEA International Press, Washington, DC.

Lu, Z., 1992. Coordination in distributed intelligence systems. MS Thesis, Report GMU/C3I-120-TH, C3I Center , George Mason University, Fairfax, VA.

Lu, Z.,, and Levis, A.H., 1992. Coordination in distributed decision making. In *Proc. 1992 IEEE International Conference on Systems, Man, and Cybernetics,* Chicago, IL.

Mesarovic, M.D., Macko, D., and Takahara, Y., 1970. *Theory of Hierarchical, Multilevel Systems*. Academic Press, New York.

Minsky, M., 1986. *The Society of Mind.* Simon and Schuster, New York.

Monguillet, J.-M., and Levis A.H., 1993. Modeling and evaluation of variable structure command and control organizations. In *Toward a Science of Command, Control, and Communications,* Jones, C., Ed., AIAA Press, Washington, DC.

Neapolitan, R., 1990. *Probabilistic Reasoning in Expert Systems: Theory and Algorithms,* John Wiley & Sons: New York.

Newell, A., and Simon, H., 1972. *Human Problem Solving* Prentice Hall, Englewood Cliffs, N. J.

Nisbett, R., and Ross, L., 1980. *Human Interface: Strategies and Shortcomings of Social Judgment* Prentice-Hall, Englewood Cliffs, N. J.

Paass, G., 1991. Integrating probabilistic rules into neural networks: A stochastic EM learning algorithm, *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pp. 264-270.

Pearl, J., 1987. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, California.

Perdu, D., Zaidi, A., Sadigh, A., Lehner, P., and Levis, A., 1993. On a methodology for team design using influence diagrams. *Proceedings of the 1993 Symposium on Command and Control Research.*, Science Applications International Corp., McLean, VA.

Peterson, J.L., 1981. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliffs, NJ.

Reisig, W., 1985. *Petri Nets, An Introduction.* Springer-Verlag, Berlin, Germany.

Remy, P., 1986. On the generation of organizational architectures using Petri Nets. MS Thesis, LIDS-TH-1630, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Remy, P., and Levis, A.H., 1988. On the generation of organizational architectures using Petri Nets. In *Advances in Petri Nets 1988, Lecture Notes in Computer Science,* Rozenberg, G., Ed., Springer-Verlag, Berlin, Germany.

Sadigh, A., 1993. Master's Thesis, in progress.

Stabile, D.A., and Levis, A.H., 1984. The design of information structures: basic allocations strategies for organizations. *Large Scale Systems,* Vol.6, No 2, North-Holland.

Tversky, A., and Kahneman, D., 1971. The belief in the 'Law of Small Numbers' *Psychological Bulletin* , **76**, pp. 105-110.

Tversky, A., and Kahneman, D., 1973. Availability: a heuristic for judging frequency and probability. *Cognitive Psychology* , **5**, pp. 207-232

Tolcott, M.A., Marvin, F.F., and Lehner, P.E., 1989. Expert decision making in evolving situations" *IEEE Transactions on Systems, Man and Cybernetics* , **SMC-19**, pp. 606-615.

Wason, P., 1960. On the failure to eliminate hypotheses in a conceptual task. *Quarterly Journal of Experimental Psychology,* **12**, pp. 129-140

Weingaertner, S.T., and Levis, A.H., 1989. Evaluation of decision aiding in submarine emergency decisionmaking" *Automatica,* Vol. **25**, No 3.